Apprendre à programmer en C pour les Nuls

Solutions des exercices

Cahier 2/3 : Partie III (chapitres 11 à 17)

Chapitre 11

ex1101

```
#include <stdio.h>

int main()
{
    int x;

    for(x=0; x<10; x++)
        puts("Ne marchez pas sur ma pelouse !");
    return(0);
}
```

Remarques

Qu'importe le nom de la variable, pourvu qu'elle soit de type `int`.

L'opérateur d'incrémentation ++ est le même que dans le livre, sauf que la boucle ne fait que dix tours.

Les accolades de la boucle autour de `puts()` étant facultatives, je les ai omises ici.

ex1102

```
#include <stdio.h>

int main()
{
    int x;

    x=0;
    while(x < 10)
    {
        puts("Ne marchez pas sur ma pelouse !");
        x++;                                        // L11
    }
    return(0);
}
```

Remarques

J'imagine que vous aviez utilisé la boucle `for` pour le premier exercice. Si vous aviez choisi `while`, utilisez `for` pour celui-ci.

L'opérateur d'incrémentation est en ligne 11.

ex1103

```
#include <stdio.h>

int main()
{
```

```
    int c;

    for(c=-5; c<5; c++)
        printf("%d ", c);          // L08
    for(; c>=-5; c--)
        printf("%d ", c);          // L10
    putchar('\n');
    return(0);
}
```

Remarques

Une boucle en C ne peut fonctionner que dans un sens, sauf à faire des acrobaties. Mieux vaut créer deux boucles successives.

L'opérateur de décrémentation est en ligne 9 dans `for`.

L'espace après le formateur `%d` en lignes 8 et 10 rend l'affichage plus lisible.

ex1104

```
#include <stdio.h>

int main()
{
    int d;

    d = -10;
    while(d < 10)
    {
        printf("%d ", d);
        d++;                       // L11
    }
    while(d >= -10)
    {
        printf("%d ", d);
        d--;                       // L16
    }
    putchar('\n');
    return(0);
}
```

Remarques

Comme dans ex1103, la première boucle `while` s'arrête quand la variable `d` vaut 10. La seconde boucle `while` n'a donc pas besoin de définir la valeur de départ de `d`.

L'opérateur d'incrémentation est en ligne 11 et celui de décrémentation en ligne 16.

La seconde boucle peut s'écrire ainsi :

```
while(d > -11)
```

Le résultat ne change pas, mais c'est moins lisible : puisque nous devons aller de -10 à 10, l'apparition de la valeur 11 peut étonner.

ex1105

```
#include <stdio.h>

int main()
{
    int a, b;
```

```
    b=16;
    printf("Avant, a n'a pas encore de valeur et b=%d\n", b);
    a = b++;                                    // L09
    printf("Apres, a=%d et b=%d\n", a, b);
    return(0);
}
```

Remarques

Voici l'affichage chez moi :

```
Avant, a n'a pas encore de valeur et b=16
```

```
Apres, a=16 et b=17
```

En ligne 9, la valeur de `b` est copiée dans `a` puis incrémentée dans un deuxième temps.

J'utilise souvent ce genre d'affichage technique pour être sûr de ce qui se passe. C'est notamment utile quand vous manipulez de spointeurs.

N.d.T. : Le premier message du livre doit bien sûr se lire comme dans la solution ci-dessus.

ex1106

```
#include <stdio.h>

int main()
{
    int a, b;

    b=16;
    printf("Avant, a n'a pas encore de valeur et b=%d\n", b);
    a = ++b;
    printf("Apres, a=%d et b=%d\n", a, b);
    return(0);
}
```

Remarques

Nous avons placé l'opérateur `++` en préfixe du nom de variable `b` en ligne 9 : la valeur est incrémentée avant de la copier dans la variable `a`.

Voici l'affichage :

```
Avant, a n'a pas encore de valeur et b=16
```

```
Apres, a=17 et b=17
```

Soyez vigilant au niveau du moment d'effet des opérateurs d'incrémentation et décrémentation, selon qu'ils sont en préfixe ou en suffixe si vous ne voulez pas avoir des surprises.

ex1107

```
#include <stdio.h>

#define VALEUR 5

int main()
{
    int a;

    printf("Modulo %d:\n",VALEUR);
    for(a=0; a<30; a++)
      {  printf("%d %% %d = %d\n", a, VALEUR, a%VALEUR); }
    return(0);
```

```
}
```

Remarques

Voici un exemple de l'affichage résultant :

```
Modulo 5 :
0 % 5 = 0
1 % 5 = 1
2 % 5 = 2
3 % 5 = 3
4 % 5 = 4
5 % 5 = 0
6 % 5 = 1
7 % 5 = 2
8 % 5 = 3
9 % 5 = 4
10 % 5 = 0
11 % 5 = 1
12 % 5 = 2
13 % 5 = 3
14 % 5 = 4
15 % 5 = 0
```

(etc.)

Si la division est entière, l'opérateur renvoie bien sûr zéro puisqu'il n'y a pas de reste.

Un emploi répandu de modulo concerne le besoin d'agir tous les *n* éléments dans une liste. Voici un exemple :

```
if(mavar%2)
{
  /* Lancer l'action si reste nul */
}
else
{
  /* Lancer l'action normale */
}
```

Ici, l'action spéciale sera lancée pour chaque valeur paire.

ex1108

```
#include <stdio.h>

#define VALEUR 3

int main()
{
    int a;

    printf("Modulo %d:\n", VALEUR);
    for(a=0; a<30; a++)
    {
        printf("%d %% %d = %d\n", a, VALEUR, a%VALEUR);
    }
    return(0);
}
```

Remarques

La seule retouche est en ligne 3, mais l'effet se propage à toutes les occurrence de la constante.

Soyez vigilant au niveau de l'ordre dans l'expression du modulo : *la plus grande valeur en premier*. SI vous inversez, le résultat ne sera pas celui prévu.

ex1109

```
#include <stdio.h>

int main()
{
    float alpha;

    alpha=501;
    printf("alpha = %.1f\n", alpha);
    alpha+=99;
    printf("alpha = %.1f\n", alpha);
    alpha-=250;
    printf("alpha = %.1f\n", alpha);
    alpha/=82;
    printf("alpha = %.1f\n", alpha);
    alpha*=4.3;
    printf("alpha = %.1f\n", alpha);
    return(0);
}
```

Remarques

Voici l'affichage résultant :

```
alpha = 501.0
alpha = 600.0
alpha = 350.0
alpha = 4.3
alpha = 18.4
```

Comme dans un exemple du Chapitre 10, nous utilisons le formateur `%.1f` dans les `printf()`. pour éviter d'encombrer l'affichage par des zéros. Le Chapitre 13 détaille les formateurs et métacaractères de la fonction `printf()`.

ex1110

```
#include <stdio.h>

int main()
{
    int cinq;

    for(cinq=5; cinq<101; cinq+=5)     // L07
        printf("%d\n", cinq);
    return(0);
}
```

Remarques

Comme déjà dit, qu'importe les noms que vous avez choisi pour les variables. Intéressons-nous au troisième membre de la condition `for` : l'opérateur `+=` fait progresser par pas de 5 (ligne 7).

Pour le deuxième membre, vous auriez pu écrire `cinq<=100`.

Félicitations si vous avez construit la boucle avec `while`. Sinon, vous pouvez le reformuler ainsi en guise de bonus.

ex1111

```
#include <stdio.h>
#include <math.h>
```

```c
int main()
{
    float resultat, valeur;

    printf("Indiquez une valeur fractionnaire : ");
    scanf("%f", &valeur);
    resultat = sqrt(valeur);
    printf("Racine carre de %.2f = %.2f\n", valeur, resultat);
    resultat = pow(valeur, 3);
    printf("%.2f puissance cubique = %.2f\n", valeur, resultat);
    resultat = floor(valeur);
    printf("Arrondi par defaut de %.2f = %.2f\n", valeur, resultat);
    resultat = ceil(valeur);
    printf("Arrondi par exces de %.2f  = %.2f\n", valeur, resultat);
    return(0);
}
```

Remarques

Le formateur `%.2f` limite l'affichage à deux décimales.

Voici un essai avec la valeur 45.3 :

```
Indiquez une valeur fractionnaire : 45.3
Racine carre de 45.30 = 6.73
45.30 puissance cubique = 92959.67
Arrondi par defaut de 45.30 = 45.00
Arrondi par exces de 45.30 = 46.00
```

ex1112

```c
#include <stdio.h>
#include <math.h>

int main()
{
    float lesdeux;

    puts("Les nombres magiques de l'informatique :");
    for(lesdeux=0; lesdeux<=10; lesdeux++)
        printf("2^%.0f = %.0f\n", lesdeux, pow(2, lesdeux));   // L10
    return(0);
}
```

Remarques

Il y a de nombreuses manières de résoudre cet exercice.

La fonction `pow()` travaille avec des flottants, donc la variable `lesdeux` est déclarée `float`.

Dans d'autres langages, l'élévation à la puissance utilise un symbole ou un opérateur, pas en C.

Cette fonction `pow()` est appelée en ligne 10 et ce qu'elle renvoie est consommé sur le champ. Vous auriez bien sûr stocker d'abord le renvoi dans une variable.

Le formateur `%.0f` évite d'afficher la partie décimale.

Voici un exemple d'exécution satisfaisant :

```
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32
2^6 = 64
```

```
2^7 = 128
2^8 = 256
2^9 = 512
2^10 = 1024
```

ex1113

```
#include <stdio.h>

int main()
{
    float degres, radians;

    printf("Indiquez un angle en degres : ");
    scanf("%f", &degres);
    radians = 0.0174532925*degres;
    printf("%.2f degres valent %.2f radians.\n", degres, radians);
    return(0);
}
```

Remarques

Voici un exemple d'exécution :

```
Indiquez un angle en degres : 150
150.00 degres valent 2,62 radians.
```

ex1114

```
#include <stdio.h>

int main()
{
    float degres, radians;

    printf("Indiquez un angle en radians : ");
    scanf("%f", &radians);
    degres = 57.2957795*radians;
    printf("%.2f radians valent %.2f degres.\n", radians, degres);
    return(0);
}
```

Remarques

Nous réalisons ici la conversion inverse de celle de l'exercice précédent.

Voici un exemple d'exécution :

```
Indiquez un angle en radians : 3.14159
3.14 radians valent 180.00 degres.
```

ex1115

```
#include <stdio.h>
#include <math.h>

#define PI             3.14159
#define AMPLITUDE     70       // Amplitude du signal
#define LONGUEURONDE  .1       // Oui, inverse de la frequence
```

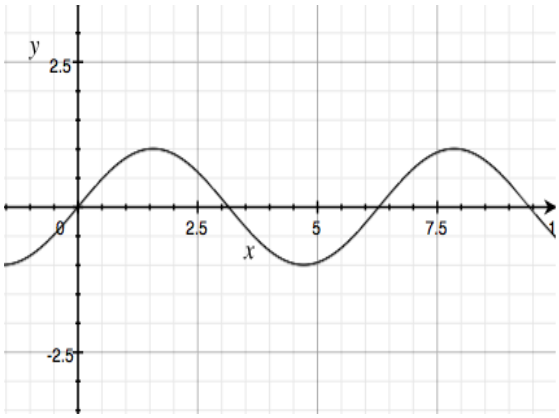
```
int main()
{
    float graph, s, x;

    for(graph=0; graph<PI; graph+=LONGUEURONDE)
    {
        s = sin(graph);
        for(x=0; x<s*AMPLITUDE; x++)
            putchar('*');
        putchar('\n');
    }
    return(0);
}
```

Remarques

N.d.T. : La version anglaise avait interverti les deux paramètres d'amplitude et de longueur d'onde. Les choses sont remises en ordre dans cette solution. Cela n'a pas d'effet sur l'intérêt pédagogique de l'exemple. L'erreur (que le traducteur n'a pas intercepté au départ) est liée au fait que dans cet affichage en mode texte, la courbe est simulée après une rotation d'un quart de tour.

Voici un graphe de sinusoïde dans l'orientation habituelle. Notre affichage texte fait progresser la courbe de haut en bas :



La constante `PI` est nécessaire parce que les fonctions du C traitent des radians, pas des degrés. En ligne 12, la boucle `for` va de zéro à `PI` radians, soit un demi-cercle.

Vous pouvez faire des essais en retouchant la valeur des constantes.

La constante `AMPLITUDE` détermine la hauteur du graphe en colonnes écran (donc de gauche à droite). Réduisez cette constante pour rendre la courbe plus plate.

La constante `LONGUEURONDE` contrôle la fréquence ou longueur d'onde. Réduisez la valeur pour que la courbe tienne dans un écran texte ; augmentez-la pour aplatir l'onde (dans le sens vertical puisque l'affichage est tourné de 90°.

ex1116

```
#include <stdio.h>
#include <math.h>

#define PI            3.14159
#define AMPLITUDE 35
#define LONGUEURONDE       .1

int main()
```

```
{
    float graph, s, x;

    for(graph=0; graph<2*PI; graph+=LONGUEURONDE)
    {
        s = cos(graph);
        s+=1.0;             /* Compenser pour valeurs négatives */
        for(x=0; x<s*AMPLITUDE; x++)
            putchar('*');
        putchar('\n');
    }
    return(0);
}
```

Remarques

La ligne commentée est une astuce. En ajoutant 1 à `s`, nous évitons les valeurs négatives qui s'afficheraient mal dans ce mode texte. Tout l'affichage est décalé. Ne réutilisez surtout pas cette astuce pour calculer des ondes ou des angles !

Aucun souci si vous n'avez pas réussi ici. J'ai moi-même mis un certain temps à être satisfait du résultat.

ex1117

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

int main()
{
    int r, a, b;

    puts("100 chiffres au hasard");
    for(a=0; a<20; a++)
    {
        for(b=0; b<5; b++)
        {
            r=rand();
            printf("%d\t", r);
        }
        putchar('\n');
    }
    return(0);
}
```

Remarques

Certains compilateurs proposent une fonction `random()` améliorée par rapport à `rand()`. Elle renvoie une valeur de type `long int` au lieu de `int`.

ex1118

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

int main()
{
    int r, a, b;

    puts("100 chiffres au hasard");
    for(a=0; a<20; a++)
    {
```

```
        for(b=0; b<5; b++)
        {
            r=rand();           // L13
            r%=21;
            printf("%d\t", r);
        }
        putchar('\n');
    }
    return(0);
}
```

Remarques

Le programme affiche cent valeurs au hasard entre 0 et 20.

La valeur zéro peut apparaître parce que l'expression `r%=21` en revoie un quand il n'y a pas e reste, donc quand la valeur de départ est divisible par 21.

Vous pouvez regrouper les lignes 13 et 14 ainsi :

```
r=rand() % 21;
```

La plupart du temps, vos fonctions de production de valeurs aléatoires vont profiter de l'opérateur modulo pour borner la plage de valeurs générées, comme ici.

ex1119

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

int main()
{
    unsigned graine;
    int r, a, b;

    printf("Indiquez une valeur int pour la graine : ");
    scanf("%u", &graine);
    srand(graine);
    for(a=0; a<20; a++)
    {
        for(b=0; b<5; b++)
        {
            r=rand();
            printf("%d\t", r);
        }
        putchar('\n');
    }
    return(0);
}
```

Remarques

Le formateur `%u` dans `scanf()` garantit que la valeur saisie est strictement positive.

ex1120

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

int main()
{
    int r, a, b;
```

```
    srand( (unsigned)time(NULL) );   // Transtypage
    for(a=0; a<20; a++)
    {
        for(b=0; b<5; b++)
        {
            r=rand();
            printf("%d\t", r);
        }
        putchar('\n');
    }
    return(0);
}
```

Remarques

La fonction `time()` en ligne 10 requiert l'importation du fichier d'en-tête `time.h`, ce qui est fait en début de fichier.

N.d.T. : Notez l'utilisation d'un transtypage en ligne 9.

ex1121

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

int main()
{
    int r, devinezMoi;

    srand( (unsigned)time(NULL) );
    r=rand();
    printf("Devinez le chiffre secret : ");
    scanf("%d", &devinezMoi);
    if(devinezMoi == r)
    {
        puts("Quels pouvoirs !");
        return(0);
    }
    if(devinezMoi != r)
    {
        printf("Ce n'est pas %d !\n", devinezMoi);
        printf("Il fallait le trouver : %d\n", r);
        return(1);
    }
}
```

Remarques

Le chiffre à deviner n'est plus figé dans une constante comme dans l'exercice 0806, mais produit au hasard par `rand()`.

Vous avez peu de chances de trouver la réponse du fait que la plage de valeurs possibles est celle du type entier `int` qui s'étend à plusieurs millions, en positif et en négatif !

Amélioration 1 : Exploitez l'opérateur modulo pour contraindre la plage de valeurs entre 1 et 10.

Amélioration 2 : Offre au joueur trois essais en plaçant la partie dialogue dans une boucle. Après chaque essai, indiquez si le chiffre est supérieur ou inférieur à sa saisie.

ex1122

```
#include <stdio.h>
```

```
int main()
{
    int resultat;

    resultat = 20-5*2+42/6;
    printf("20-5*2+42/6=%d\n", resultat);
    return(0);
}
```

Remarques

Vous auriez pu vous contenter d'afficher le résultat, mais rappeler l'équation n'est pas inutile.

Voici l'affichage :

`20-5*2+42/6=17`

Les deux opérations `5*2` et `42/6` sont traitées en premier, ce qui donne 20-10+7, qu se lit de gauche à droite sans ambiguïté pour donner 17.

ex1123

```
#include <stdio.h>

int main()
{
    int resultat;

    resultat = 12/3/2;
    printf("12/3/2=%d\n", resultat);
    return(0);
}
```

Remarques

Aucun piège dans cette expression puisque la division est évaluée de gauche à droite.

Si nous ajoutons des multiplication, comme dans `2 * 12 / 4 * 5`, l'évaluation reste naturelle car division et multiplication ont la même priorité.

Pouvez-vous deviner ce que donne `2 * 12 / 4 * 5` ? Vérifiez en écrivant l'équation dans le code source.

ex1124

```
#include <stdio.h>

int main()
{
    int resultat;

    resultat = (12-5)*2;
    printf("(12-5)*2=%d\n", resultat);
    return(0);
}
```

Remarque

Nous avons délimité `12-5` par des parenthèses pour que la soustraction soit faire en premier, avant la multiplication par 2.

ex1125

```
#include <stdio.h>
```

```
int main()
{
    int resultat;

    resultat = (20-5)*(2+42)/6;
    printf("(20-5)*(2+42)/6=%d\n", resultat);
    return(0);
}
```

Remarque

Voici l'affichage résultant :

```
(20-5)*(2+42)/6=110
```

Chapitre 12

ex1201

```
#include <stdio.h>

int main()
{
    int scoremax1, scoremax2, scoremax3;

    printf("Votre meilleur score : ");
    scanf("%d", &scoremax1);
    printf("Second meilleur score : ");
    scanf("%d", &scoremax2);
    printf("Troisieme meilleur score : ");
    scanf("%d", &scoremax3);

    puts("Voici vos meilleurs scores :");
    printf("#1 %d\n", scoremax1);
    printf("#2 %d\n", scoremax2);
    printf("#3 %d\n", scoremax3);

    return(0);
}
```

Remarques

Voici l'affichage résultant, avec les saisies en gras :

```
Votre meilleur score : 750
Second meilleur score : 699
Troisieme meilleur score : 675
Voici vos meilleurs scores :
#1 750
#2 699
#3 675
```

Vous avez fait mieux que moi.

ex1202

```
#include <stdio.h>
```

```
int main()
{
    int scoremax1, scoremax2, scoremax3,scoremax4;

    printf("Votre meilleur score : ");
    scanf("%d", &scoremax1);
    printf("Second meilleur score : ");
    scanf("%d", &scoremax2);
    printf("Troisieme meilleur score : ");
    scanf("%d", &scoremax3);
    printf("Quatrieme meilleur score : ");
    scanf("%d", &scoremax4);

    puts("Voici vos meilleurs scores :");
    printf("#1 %d\n", scoremax1);
    printf("#2 %d\n", scoremax2);
    printf("#3 %d\n", scoremax3);
    printf("#4 %d\n", scoremax4);

    return(0);
}
```

Remarque

Cet exemple fonctionne, mais pas très bien.

ex1203

```
#include <stdio.h>

int main()
{
    int scoremax[4];                    // L05

    printf("Votre meilleur score : ");
    scanf("%d",&scoremax[0]);
    printf("Second meilleur score : ");
    scanf("%d",&scoremax[1]);
    printf("Troisieme meilleur score : ");
    scanf("%d",&scoremax[2]);
    printf("Quatrieme meilleur score : ");
    scanf("%d",&scoremax[3]);

    puts("Voici vos meilleurs scores :");
    printf("#1 %d\n",scoremax[0]);
    printf("#2 %d\n",scoremax[1]);
    printf("#3 %d\n",scoremax[2]);
    printf("#4 %d\n",scoremax[3]);

    return(0);
}
```

Remarques

Le tableau est déclaré en ligne 5 sous le nom `scoremax` et prêt à recevoir quatre éléments.

Avec `scanf()`, nous peuplons le tableau en désignant les quatre éléments par les indices `0` à `3`.

Les lignes 17 à 20 affichent le contenu de `scoremax`, soit les éléments `0` à `3`. Chaque élément est une variable simple de type `int`.

ex1204

```
#include <stdio.h>

int main()
{
    int scoremax[4];
    int x;

    for(x=0; x<4; x++)
    {
        printf("Votre score #%d : ", x+1);    // L10
        scanf("%d", &scoremax[x]);
    }

    puts("Voici vos meilleurs scores :");
    for(x=0; x<4; x++)
        printf("#%d %d\n", x+1, scoremax[x]);

    return(0);
}
```

Remarques

L'astuce `x+1` en ligne 10 permet de ne pas perturber la numérotation des indices de tableau tout en affichant un numéro d'élément moins bizarre, car commençant à un, pas à zéro. Vous utiliserez souvent ce correctif d'humanisation.

L'opérateur `&` est obligatoire dans `scanf()` pour que la fonction sache à quelle adresse stocker la valeur lue. Cet opérateur n'est pas concerné par le fait que la variable en cours fait partie d'un tableau ou pas.

ex1205

```
#include <stdio.h>

int main()
{
    int scoremax[10];
    int x;

    for(x=0;x<10;x++)
    {
        printf("Votre score #%d : ", x+1);
        scanf("%d",&scoremax[x]);
    }

    puts("Voici vos meilleurs scores :");
    for(x=0; x<10; x++)
        printf("#%d %d\n", x+1, scoremax[x]);

    return(0);
}
```

Remarques

Et un point de bonus de bonus *Pour les nuls* si vous aviez prévu une constante pour la taille du tableau et pour la variable de boucle dans les deux boucles `for`. Le métier semble bien rentrer !

Le compilateur ne se plaint pas si vous oubliez d'agrandir le tableau dans la déclaration. Si vous déclarez `scoremax[3]`, les éléments suivants vont être stockés en dehors de la zone mémoire du tableau, écrasant la valeur d'autres variables et mettant en péril le fonctionnement. C'est au

programmeur de faire attention que ses tableaux soient prévus assez grands pour recevoir tout ce qu'il y stocke ensuite.

ex1206

```
#include <stdio.h>

int main()
{
    float tabBourseFin[] = { 14450.06, 14458.62, 14539.14, 14514.11, 14452.06 };
    int jourCota;

    puts("Cours de fermeture bourse");
    for(jourCota=0; jourCota<5; jourCota++)
        printf("Jour %d: %.2f\n", jourCota+1, tabBourseFin[jourCota]);
    return(0);
}
```

Remarques

Pour plus de lisibilité, vous pouvez énoncer le valeurs des éléments sur des ligne distinctes :

```
float tabBourseFin[] = {
  14450.06,
  14458.62,
  14539.14,
  14514.11,
  14452.06 };
```

L'accolade fermante sera encore plus visible sur une ligne à part, mais **n'oubliez pas d'oublier** (oui, oui) la virgule après le dernier élément (sinon, le compilateur va croire qu'il manque encore un élément) :

```
float tabBourseFin[] = {
  14450.06,
  14458.62,
  14539.14,
  14514.11,
  14452.06
};
```

Certains structurent l'apparence des valeurs initiales des éléments en lignes et colonnes, même si c'est purement esthétique :

```
float tabBourseFin[] = {
  14450.06, 14458.62, 14539.14,
  14514.11, 14452.06
};
```

Dans l'instruction `printf()` du code, nous retrouvons l'astuce `jourCota+1` pour afficher les numéros d'éléments en commençant à un.

ex1207

```
#include <stdio.h>
#include <math.h>

int main()
{
    int   tabint1[] = { 10, 12, 14, 15, 16, 18, 20 };  // L06
    float tabflo2[7];
    int x;
```

```
    for(x=0; x<7; x++)                              // L10
        tabflo2[x] = sqrt(tabint1[x]);

    for(x=0; x<7; x++)
        printf("Racine carree de %d = %.2f\n", tabint1[x], tabflo2[x]);
    return(0);                                      // L14
}
```

Remarques

En ligne 6, nous répondons au premier objectif : déclarer un tableau de sept valeurs entières spécifiques.

En ligne 7, nous créons un tableau vide, cette fois de type `float` car l'énoncé demande d'y stocker des racines carrées qui sont rarement entières. Si vous avez omis cette nuance, vous vous en rendez compte vite devant les étranges résultats affichés.

Alors que d'autres langage sont plus pointilleux à ce sujet, le C ne se plaint pas quand il vous voit demander de stocker dans une valeur `int` la valeur flottante que renvoie la fonction `sqrt()`. Mais les valeurs sont erronées, comme dit plus haut.

Les lignes 10 et 11 font l'essentiel du travail noble : extraire la racine carrée d'un élément du premier tableau et la stocker dans le second, dans une boucle `for`. L'extraction en ligne 11 exploite la fonction `sqrt()` qui réclame l'insertion du fichier `math.h` en début de code.

Une autre boucle en lignes 13 et 14 affiche les résultats. Vous pouvez vous être limité à présenter les valeurs successives, au lieu de prévoir une légende pour chacune comme moi.

ex1208

```
#include <stdio.h>

int main()
{
    char phrase[] = "Texte insignifiant";
    int index;

    index = 0;
    while(phrase[index] != '\0')
    {
        putchar(phrase[index]);
        index++;
    }
    putchar('\n');
    return(0);
}
```

ex1209

```
#include <stdio.h>

int main()
{
    char prenom[16];

    printf("Quel est votre prenom ? ");
    fgets(prenom, 16, stdin);
    printf("Ravi de vous rencontrer, %s\n", prenom);
    return(0);
}
```

Remarque

En ligne 15, nous lisons l'entrée standard avec `fgets()` en lui demandant de n'accepter que les quinze premiers caractères saisis, auquel est ajouté le zéro terminal `\0`. Les tableaux C ne savent pas se protéger des débordements d'écriture et de lecture (après le dernier élément). Pour éviter ce genre de débordement, nous profitons de la limite que nous pouvons demander à `fgets()`.

ex1210

```
#include <stdio.h>

int main()
{
    char prenom[16], nomfam[16];

    printf("Quel est votre prenom ? ");
    fgets(prenom, 16, stdin);
    printf("Quel est votre nom ? ");
    fgets(nomfam, 16, stdin);
    printf("Bienvenue, cher %s %s\n", prenom, nomfam);
    return(0);
}
```

Remarques

Vous pouvez bien sûr prévoir plus grand pour le tableau `nomfam[]` si vous avez des nobles ou des malgaches (*) dans vos connaissances. Dans ce cas, pensez à changer aussi le plafond de saisir de `fgets()`.

* N.d.T. : ou si vous connaissez un physicien avec un nom à particule.

ex1211

```
#include <stdio.h>

int main()
{
    char prenom[16], nomfam[16];

    printf("Quel est votre prenom ? ");
    scanf("%s", prenom);
    printf("Quel est votre nom ? ");
    scanf("%s", nomfam);
    printf("Bienvenue, cher %s %s\n",prenom, nomfam);
    return(0);
}
```

Remarques

Exceptionnellement, le préfixe d'adresse `&` n'est pas requis dans `scanf()` si les données sont envoyées dans un tableau de type `char`.

Il n'y a plus de caractère de saut de ligne parasite car `scanf()` s'arrête au premier espace.

ex1212

```
#include <stdio.h>

int main()
{
    char prenom[16], nomfam[16];
```

```
    printf("Quel est votre prenom ? ");
    scanf("%15s", prenom);
    printf("Quel est votre nom ? ");
    scanf("%15s", nomfam);
    printf("Bienvenue, cher %s %s\n", prenom, nomfam);
    return(0);
}
```

Remarque

L'arrêt de lecture par `scanf()` au premier espace est gênant dans les noms composés avec espace.
C'est un souci de `scanf()` utilisé en saisie de données.

ex1213

```
#include <stdio.h>

#define TAILLE 6

int main()
{
    int tabulles[] = { 95, 60, 6, 87, 50, 24 };
    int interne, externe, temp, x;

/* Affiche le tableau original */
    puts("Tableau de depart :");
    for(x=0; x<TAILLE; x++)
        printf("%d\t", tabulles[x]);
    putchar('\n');

/* Tri de tabulles */
    for(externe=0; externe<TAILLE-1; externe++)
    {
        for(interne=externe+1; interne<TAILLE; interne++)
        {
            if(tabulles[externe] > tabulles[interne])
            {
                temp = tabulles[externe];
                tabulles[externe] = tabulles[interne];
                tabulles[interne] = temp;
            }
        }
    }

/* Affiche le tableau apres tri */
    puts("Tableau apres tri :");
    for(x=0; x<TAILLE; x++)
        printf("%d\t", tabulles[x]);
    putchar('\n');

    return(0);
}
```

ex1214

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

#define TAILLE 40
```

```
int main()
{
    int tabulles[40];
    int interne, externe, temp, x;

    srand((unsigned)time(NULL));           // L12
/* Affiche le tableau initial */
    puts("Tableau de depart :");
    for(x=0; x<TAILLE; x++)
    {
        tabulles[x] = rand();              // L17
        tabulles[x] = (tabulles[x] % 100)+1;
        printf("%d\t", tabulles[x]);
    }
    putchar('\n');

/* Tri de tabulles */
    for(externe=0; externe<TAILLE-1; externe++)
    {
        for(interne=externe+1; interne<TAILLE; interne++)
        {
            if(tabulles[externe] > tabulles[interne])
            {
                temp = tabulles[externe];
                tabulles[externe] = tabulles[interne];
                tabulles[interne] = temp;
            }
        }
    }

/* Affiche le tableau apres tri */
    puts("Tableau apres tri :");
    for(x=0; x<TAILLE; x++)
        printf("%d\t", tabulles[x]);
    putchar('\n');

    return(0);
}
```

Remarques

Il faut ajouter les directives d'inclusion pour `stdlib.h` et `time.h` pour pouvoir appeler les fonctions `rand()`, `srand()` et `time()`.

Nous ensemençons le générateur aléatoire en ligne 12 (revoir le Chapitre 11).

Les lignes 17 et 18 écrivent les valeurs dans le tableau, en deux lignes pour être plus lisible. Nous stockons la valeur générée en ligne 17 puis nous la réduisons à la plage 1 à 100 par modulo une fois qu'elle est dans le tableau. Vous auriez pu passer par l'étape d'une variable de travail `int` ou même appeler directement la fonction `rand()` dans une seule instruction. Qu'importe le moyen, pourvu que le résultat soit là.

ex1215

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

#define TAILLE 40

int main()
{
```

```
    int tabulles[40];
    int interne, externe, temp, x;

    srand((unsigned)time(NULL));
/* Affiche le tableau initial */
    puts("Tableau de depart :");
    for(x=0; x<TAILLE; x++)
    {
        tabulles[x] = rand();
        tabulles[x] = (tabulles[x] % 100)+1;
        printf("%d\t", tabulles[x]);
    }
    putchar('\n');

/* Tri de tabulles */
    for(externe=0; externe<TAILLE-1; externe++)
    {
        for(interne=externe+1; interne<TAILLE; interne++)
        {
            if(tabulles[externe] < tabulles[interne])   // L28
            {
                temp = tabulles[externe];               // L30
                tabulles[externe] = tabulles[interne];
                tabulles[interne] = temp;
            }
        }
    }

/* Affiche le tableau apres tri */
    puts("Tableau final :");
    for(x=0; x<TAILLE; x++)
        printf("%d\t", tabulles[x]);
    putchar('\n');

    return(0);
}
```

Remarque

Le tri inverse se résume à remplacer l'opérateur > par < en ligne 28. Vous pourriez aussi inverser la séquence de permutation des éléments en lignes 30 à 32, mais c'est facultatif. Les éléments seront triés comme demandé.

ex1216

```
#include <stdio.h>

#define TAILLE 19

int main()
{
    char tabulles[] = "Vive le langage C !";
    char temp;
    int interne,externe,x;

/* Affiche le tableau initial */
    puts("Tableau de depart :");
    for(x=0; x<TAILLE; x++)
        printf("%c", tabulles[x]);
    putchar('\n');
```

```
/* Tri de tabulles */
    for(externe=0; externe<TAILLE-1; externe++)
    {
        for(interne=externe+1; interne<TAILLE; interne++)
        {
            if(tabulles[externe] > tabulles[interne])
            {
                temp = tabulles[externe];
                tabulles[externe] = tabulles[interne];
                tabulles[interne] = temp;
            }
        }
    }

/* Affiche le tableau apres tri */
    puts("Tableau final :");
    for(x=0; x<TAILLE; x++)
        printf("%c", tabulles[x]);
    putchar('\n');

    return(0);
}
```

Remarques

Le tri ne dépend pas du type de valeur des variables.

Voici un exemple d'affichage :

```
Tableau de depart :
Vive le langage C !
Tableau final :
    !CVaaeeeggillnv
```

Les codes ASCII des majuscules sont inférieurs à ceux des minuscules et l'espace leur est encore inférieur, d'où la série d'espaces en début de chaîne traitée.

ex1217

```
#include <stdio.h>

int main()
{
    char tictactoe[3][3];
    int x, y;

/* Initialise la matrice */
    for(x=0; x<3; x++)
        for(y=0 ;y<3; y++)
            tictactoe[x][y] = '.';
    tictactoe[1][1] = 'X';

/* Affiche le plateau de jeu */
    puts("Une partie de Tic-Tac-Toe ?");
    for(x=0; x<3; x++)
    {
        for(y=0; y<3; y++)
            printf("%c  ", tictactoe[x][y]);
        putchar('\n');
    }
    return(0);
}
```

ex1218

```
#include <stdio.h>

#define TAILLE 3

int main()
{
    char president[TAILLE][8] = {              // L07
        "Sarkozy",
        "Coty",
        "Grevy"
    };
    int x, index;

    for(x=0; x<TAILLE; x++)
    {
        index = 0;
        while(president[x][index] != '\0')    // L17
        {
            putchar(president[x][index]);
            index++;
        }
        putchar('\n');
    }
    return(0);
}
```

Remarques

La boucle `while` (ligne 17) peut vous étonner à cause des crochets pour désigner chaque élément (un caractère) de la seconde dimension du tableau : `president[x][index]`. En effet, `president[x]` désigne le début d'une ligne, donc ici d'une chaîne contenant un nom. La mention `[index]` permet d'avancer dans cette ligne. L'expression complète vise un caractère et il n'est pas inutile de voir comment les niveaux s'articulent.

Notez le mode d'initialisation du tableau de type `char` en ligne 7 (jusqu'à 10).

ex1219

```
#include <stdio.h>

#define TAILLE 3

int main()
{
    char president[TAILLE][8] = {
        "Sarkozy",
        "Coty",
        "Grevy"
    };
    int x, index;

    for(x=0; x<TAILLE; x++)
        puts(president[x]);
    return(0);
}
```

Remarque

L'écriture `president[x]` désigne un élément qui en contient huit de type `char`, donc une chaîne. C'est le seul cas d'utilisation autorisé de cette désignation, lié au fait qu'en langage C, une chaîne est en fait un tableau de caractères. Mais ne vous risquez pas à faire de même avec un tableau d'entiers `int` !

ex1220

```
#include <stdio.h>

#define TAILLE 6

int main()
{
    char president[TAILLE][11] = {
        "Mitterrand",
        "Chirac",
        "Pompidou",
        "Sarkozy",
        "Coty",
        "Grevy"
    };
    int x, index;

    for(x=0; x<TAILLE; x++)
        puts(president[x]);
    return(0);
}
```

Remarques

Vous avez changé la valeur de `TAILLE` en ligne 3 ?

Chaque chaîne a été rendue plus spacieuse pour pouvoir y stocker les neuf lettres de Mitterand et le zéro terminal.

ex1221

```
#include <stdio.h>

int main()
{
    char tictactoe[3][3][3];
    int x,y,z;

/* Initialisation */
    for(x=0; x<3; x++)
        for(y=0; y<3; y++)
            for(z=0; z<3; z++)
                tictactoe[x][y][z]='.';
    tictactoe[1][1][1] = 'X';

/* Affichage plateau */
    puts("C'est parti pour un Tic-Tac-Toe en 3D ?");
    for(z=0; z<3; z++)
    {
        printf("Niveau %d\n", z+1);
        for(x=0; x<3; x++)
        {
            for(y=0; y<3; y++)
                printf("%c  ", tictactoe[x][y][z]);
            putchar('\n');
```

```
        }
    }
    return(0);
}
```

ex1222

```
#include <stdio.h>

int main()
{
    char tictactoe[3][3] = {
        '.', '.', '.',
        '.', 'X', '.',
        '.', '.', '.'
    };
    int x,y;

    puts("On joue au Tic-Tac-Toe en 3D ?");
    for(x=0; x<3; x++)
    {
        for(y=0; y<3; y++)
            printf("%c  ", tictactoe[x][y]);
        putchar('\n');
    }
    return(0);
}
```

Remarques

Cet exemple montre qu'il est parfois plus efficace de peupler le tableau de façon statique au lieu d'écrire des instructions de remplissage. Mais lorsque les dimensions deviennent importantes, comme pour un plateau d'échecs, je reviens à la méthode dynamique par instruction.

L'atelier Code::Blocks va peut-être émettre un message comme quoi il manque des accolades, mais vous pouvez l'ignorer car il active tous les avertissements, y compris le *-Wmissing-braces*. Il ne manque pas d'accolades ! (Un exemple de l'excès de zèle des compilateurs avec lesquels il faut bien vivre.)

ex1223

```
#include <stdio.h>

#define TAILLE 5

void afficherTablo(int tablo[]);       // L05

int main()
{
    int n[] = { 1, 2, 3, 5, 7 };

    puts("Votre tableau :");
    afficherTablo(n);
    return(0);
}

void afficherTablo(int tablo[])       // L16
{
    int x;

    for(x=0; x<TAILLE; x++)
        printf("%d\t", tablo[x]);
    putchar('\n');
```

```
}
```

Remarques

Les arguments de fonction de type tableau ne doivent pas obligatoirement porter le même nom que la variable qui va être transmise (ligne 5). Vous pouvez même écrire ceci :

```
void afficherTablo(int[]);
```

Cette déclaration informe le compilateur que la fonction `afficherTablo()` doit recevoir en entrée un seul argument de type tableau d'entiers. C'est parfait ! Il reste à donner un nom de tableau dans la déclaration de début de corps de la fonction (ligne 16).

ex1224

```
#include <stdio.h>

#define TAILLE 5

void afficherTablo(int tablo[]);
void incremTablo(int tablo[]);

int main()
{
    int n[] = { 1, 2, 3, 5, 7 };

    puts("Tableau de depart :");
    afficherTablo(n);
    incremTablo(n);
    puts("Apres appel de incremTablo() :");
    afficherTablo(n);
    return(0);
}

void afficherTablo(int tablo[])
{
    int x;

    for(x=0; x<TAILLE; x++)
        printf("%d\t",tablo[x]);
    putchar('\n');
}

void incremTablo(int tablo[])
{
    int x;

    for(x=0; x<TAILLE; x++)
        tablo[x]++;                    // L34
}
```

Remarques

En ligne 34, l'opérateur d'incrémentation `++` sert à augmenter de un la valeur que contient l'élément de tableau (pas son indice). Nuos aurions pu utiliser `+=`, comme dans `tablo[x]+=1`, mais `++` est parfait ici.

Exemple d'affichage résultant :

```
Tableau de depart :
1    2    3    5    7
Apres appel de incremTablo() :
2    3    4    6    8
```

Chapitre 13

ex1301

```
#include <stdio.h>
#include <ctype.h>

int main()
{
    char phrase[] = "Art. 4. : La liberté consiste à pouvoir faire tout ce qui ne
    nuit pas à autrui : ainsi, l'exercice des droits naturels de chaque homme n'a
    de bornes que celles qui assurent aux autres Membres de la Société la
    jouissance de ces mêmes droits. Ces bornes ne peuvent être déterminées que
    par la Loi. ";

    int index;
    int alpha, blank, punct;

    alpha = blank = punct = 0;

/* Collecte */
    index = 0;
    while(phrase[index])
    {
        if(isalpha(phrase[index]))
            alpha++;
        if(isblank(phrase[index]))
            blank++;
        if(ispunct(phrase[index]))
            punct++;
        index++;
    }

/* Affichage */
    printf("\"%s\"\n", phrase);
    puts("Statistiques :");
    printf("%d lettres de l'alphabet\n", alpha);
    printf("%d blancs\n", blank);
    printf("%d signes de ponctuation\n", punct);

    return(0);
}
```

Remarques

* If you copy and paste this text into your editor, ensure that the `phrase[]` variable at ligne 6 appears as a single line. Do not break it up! I've split the line above so that it fits well on this web page. Otherwise, it would be about three feet long and make the page all ugly.

* It's the `index++;` statement at ligne 21 that moves the program through each character in the string.

* The *printf()* statement at ligne 26 contains a few escape sequences. The `\"` is required to stick a double quote in the output, which quotes the string stored in `phrase[]`.

ex1302

```
#include <stdio.h>
#include <ctype.h>

int main()
{
    char phrase[] = "Art. 4. : La liberté consiste à pouvoir faire tout ce qui ne
```

```
    nuit pas à autrui : ainsi, l'exercice des droits naturels de chaque homme n'a
    de bornes que celles qui assurent aux autres Membres de la Société la
    jouissance de ces mêmes droits. Ces bornes ne peuvent être déterminées que
    par la Loi. ";

    int index;
    int alpha,blank,lower,punct,upper;

    alpha = blank = punct = lower = upper = 0;

/* Collecte */
    index = 0;
    while(phrase[index])
    {
        if(isalpha(phrase[index]))
            alpha++;
        if(isblank(phrase[index]))
            blank++;
        if(ispunct(phrase[index]))
            punct++;
        if(islower(phrase[index]))
            lower++;
        if(isupper(phrase[index]))
            upper++;
        index++;
    }

/* Affichage */
    printf("\"%s\"\n", phrase);
    puts("Statistiques :");
    printf("%d lettres de l'alphabet\n", alpha);
    printf("%d blancs\n", blank);
    printf("%d signes de ponctuation\n", punct);
    printf("%d minuscules\n", lower);
    printf("%d MAJUSCULES\n", upper);

    return(0);
}
```

Remarques

* See my solution for exercice 13-1. The same comments apply to this solution.

ex1303

```
#include <stdio.h>
#include <ctype.h>

int main()
{
    char phrase[] = "Art. 4. : La liberté consiste à pouvoir faire tout ce qui ne
    nuit pas à autrui : ainsi, l'exercice des droits naturels de chaque homme n'a
    de bornes que celles qui assurent aux autres Membres de la Société la
    jouissance de ces mêmes droits. Ces bornes ne peuvent être déterminées que
    par la Loi. ";

    int index;
    int alpha,blank,lower,punct,upper;

    alpha = blank = punct = lower = upper = 0;

/* Collecte */
```

```
    index = 0;
    while(phrase[index])
    {
        if(isalpha(phrase[index]))
            alpha++;
        if(isblank(phrase[index]))
            blank++;
        if(ispunct(phrase[index]))
            punct++;
        if(islower(phrase[index]))
            lower++;
        if(isupper(phrase[index]))
            upper++;
        index++;
    }

/* Affichage */
    printf("\"%s\"\n", phrase);
    puts("Statistiques :");
    printf("%d lettres de l'alphabet\n", alpha);
    printf("%d blancs\n", blank);
    printf("%d signes de ponctuation\n", punct);
    printf("%d minuscules\n", lower);
    printf("%d MAJUSCULES\n", upper);
    printf("%d au total.\n", index);

    return(0);
}
```

Remarques

En fin de boucle `while`, la variable `index` contient l'indice du dernier caractère du texte, qui donne directement le nombre de caractères !

Vous auriez pu recourir à `isprint()` pour compter les caractères en écartant le zéro terminal, mais `index` fait déjà l'affaire.

ex1304

```
#include <stdio.h>
#include <ctype.h>

int main()
{
    char reponse;

    printf("Voulez-vous faire exploser la lune (O/N) ? ");
    scanf("%c", &reponse);
    reponse = toupper(reponse);
    if(reponse == 'O')
        puts("BOOM!");
    else
        puts("La lune ne craint rien.");
    return(0);
}
```

ex1305

```
#include <stdio.h>
#include <ctype.h>

int main()
{
```

```
    char reponse;

    printf("Voulez-vous faire exploser la lune (O/N) ? ");
    scanf("%c", &reponse);
    reponse = toupper(reponse);
    if(reponse == 'O')
        puts("BOOM!");
    else if(reponse == 'N')
        puts("La lune ne craint rien.");
    else
        puts("Exister, c'est choisir !");
    return(0);
}
```

Remarques

La solution évidente est une structure conditionnelle en trois branches `if/else if/else`. Un bonus pour ceux qui auront préféré l'approche basée sur `switch case`.

Si vous ne l'avez pas fait, vous pouvez reformuler votre solution avec `switch`.

ex1306

```
#include <stdio.h>
#include <ctype.h>

int main()
{
    char textecahot[] = "CeCi esT vRaiMent cHaOtiQUe !";
    int index;
    char c;

    printf("Original : %s\n", textecahot);
    index=0;
    while(textecahot[index])                // L12
    {
        c = textecahot[index];              // L14
        if(islower(c))
            textecahot[index] = toupper(c); // L16
        else if(isupper(c))
            textecahot[index] = tolower(c);
        else
            textecahot[index] = c;
        index++;
    }
    printf("Est-ce plus lisible : %s ?\n", textecahot);
    return(0);
}
```

Remarques

Classiquement, la boucle en ligne 12 scrute le texte jusqu'au zéro terminal de fin de chaîne.

J'affecte le caractère trouvé dans `textecahot[index]` à la variable `c` en ligne 14 afin de simplifier la suite, mais vous pouvez réutiliser directement `textecahot[index]`. Le même nom de variable peut apparaître des deux côtés du signe égal d'une affectation. Voyez ce remplacement de la ligne 16 :

`textecahot[index] = toupper(textecahot[index]);`

Cet exemple se fonde sur les fonctions de test `islower()` et `isupper()` pour tester chaque caractère en combinaison croisée avec `toupper()` et `tolower()` pour inverser la casse des lettres.

Les espèces de caractères sont différenciées par la structure `if/else if/else`. Si un caractère n'est pas en bas de casse (minuscule, ligne 15), soit c'est une majuscule, soit c'est un signe non-alphabétique. La ligne 17 sélectionne les MAJUSCULES. Le reste est recopié tel quel (ligne 20).

ex1307

```
#include <stdio.h>
#include <string.h>

int main()
{
    char motpasse[] = "taco";
    char tabsaisie[15];
    int match;

    printf("Le mot de passe ? ");
    scanf("%s", tabsaisie);

    match = strcmp(tabsaisie, motpasse);
    if(match == 0)
        puts("Mot de passe correct.");
    else
        puts("Mauvais mot de passe. Alertez la DGSE.");

    return(0);
}
```

Remarque

L'essentiel lorsque vous comparez des chaînes en C est de ne pas oublier que la fonction `strcmp()` et ses collègues ne renvoient pas une valeur logique TRUE/FALSE, mais zéro en cas de succès (chaînes identiques). Voyez en ligne 14.

ex1308

```
#include <stdio.h>
#include <string.h>

int main()
{
    char motpasse[] = "taco";
    char tabsaisie[15];

    printf("Le mot de passe ? ");
    scanf("%s", tabsaisie);

    if(strcmp(tabsaisie, motpasse)==0)
        puts("Mot de passe correct.");
    else
        puts("Mauvais mot de passe. Alertez la DGSE.");

    return(0);
}
```

Remarques

Sans la variable de test `match`, vous devez appeler `strcmp()` dans la condition du `if` en ligne 12.

Et un bonus pour vous si vous avez écrit votre condition comme ceci :

```
if(!strcmp(tabsaisie, motpasse))
```

L'opérateur `!` en préfixe signifie NON. Il inverse l'état logique renvoyé par `strcmp()`. Si les deux chaînes sont identiques, elle renvoie zéro, mais `!` l'opérateur inverse l'état pour satisfaire à la condition d'exécution du bloc `if`.

ex1309

```
#include <stdio.h>
#include <string.h>

int main()
{
    char motpasse[] = "taco";
    char tabsaisie[15];

    printf("Le mot de passe ? ");
    scanf("%s", tabsaisie);

    if(strcasecmp(tabsaisie, motpasse) == 0)
        puts("Mot de passe correct.");
    else
        puts("Mauvais mot de passe. Alertez la DGSE.");

    return(0);
}
```

ex1310

```
#include <stdio.h>
#include <string.h>

int main()
{
    char primo[40];
    char secundo[20];

    printf("Indiquez votre prenom : ");
    scanf("%s", primo);
    printf("Et votre nom de famille : ");
    scanf("%s", secundo);
    strcat(primo, secundo);
    printf("Ravi de vous voir, %s!\n", primo); // L14
    return(0);
}
```

Remarques

La chaîne `primo` est affichée seule en ligne 14, car elle contient à ce moment les deux chaînes mises bout à bout.

La fonction `strcat()` copie la seconde chaîne à la fin de la première. Il est donc essentiel que celle-ci offre assez d'espace libre car la fonction ne vérifie nullement si elle écrit dans l'espace de la chaîne ou pas. Méfiez-vous !

Vous pouvez vous servir de `malloc()` (décrite dans le Chapitre 20) pour réserver de l'espace mémoire à la volée pour des tampons de stockage de chaînes. C'est très pratique lorsque vous devez rabouter des chaînes sans connaître leur longueur au départ.

ex1311

```
#include <stdio.h>
#include <string.h>

int main()
{
    char primo[40];
    char secundo[20];
```

```
    printf("Indiquez votre prenom : ");
    scanf("%s", primo);
    printf("Et votre nom de famille : ");
    scanf("%s", secundo);
    strcat(primo," ");
    strcat(primo, secundo);
    printf("Ravi de vous voir, %s!\n", primo);
    return(0);
}
```

Remarques

La solution consiste à ajouter avec `strcat()` l'espace en fin de chaîne `primo` avant d'y ajouter `secundo`.

L'espace n'occupe qu'un caractère, mais il est spécifié entre guillemets comme une chaîne de longueur 1, car le compilateur attend une chaîne dans cette fonction (ligne 13).

ex1312

```
#include <stdio.h>

int main()
{
    float valfrac1 = 34.5;
    float valfrac2 = 12.3456789;

    printf("%%9.1f = %9.1f\n", valfrac1);
    printf("%%8.1f = %8.1f\n", valfrac1);
    printf("%%7.1f = %7.1f\n", valfrac1);
    printf("%%6.1f = %6.1f\n", valfrac1);
    printf("%%5.1f = %5.1f\n", valfrac1);
    printf("%%4.1f = %4.1f\n", valfrac1);
    printf("%%3.1f = %3.1f\n", valfrac1);
    printf("%%2.1f = %2.1f\n", valfrac1);
    printf("%%1.1f = %1.1f\n", valfrac1);
    printf("%%9.1f = %9.1f\n", valfrac2);
    printf("%%9.2f = %9.2f\n", valfrac2);
    printf("%%9.3f = %9.3f\n", valfrac2);
    printf("%%9.4f = %9.4f\n", valfrac2);
    printf("%%9.5f = %9.5f\n", valfrac2);
    printf("%%9.6f = %9.6f\n", valfrac2);
    printf("%%9.7f = %9.7f\n", valfrac2);
    printf("%%9.8f = %9.8f\n", valfrac2);
    return(0);
}
```

Remarque

Cet exercice permet de comprendre comment se comportent les affichages contrôlés des valeurs à virgule flottante. Vous verrez sans doute des chiffres parasites de remplissage du côté droit, produites par la machine. Elles sont liées à un affichage dépassant la simple précision du type `float` et n'ont aucune signification.

ex1313

```
#include <stdio.h>

int main()
{
    printf("%%15s = %15s\n", "hello");
    printf("%%14s = %14s\n", "hello");
    printf("%%13s = %13s\n", "hello");
    printf("%%12s = %12s\n", "hello");
```

```
    printf("%%11s = %11s\n", "hello");
    printf("%%10s = %10s\n", "hello");
    printf(" %%9s =  %9s\n", "hello");
    printf(" %%8s =  %8s\n", "hello");
    printf(" %%7s =  %7s\n", "hello");
    printf(" %%6s =  %6s\n", "hello");
    printf(" %%5s =  %5s\n", "hello");
    printf(" %%4s =  %4s\n", "hello");
    return(0);
}
```

Remarque

J'utilise une indication de largeur avec `%s` pour aligner les chaînes afin que l'affichage soit plus agréable.

ex1314

```
#include <stdio.h>
#include <ctype.h>

int main()
{
    char phrase[] = "Art. 4. : La liberté consiste à pouvoir faire tout ce qui ne
    nuit pas à autrui : ainsi, l'exercice des droits naturels de chaque homme n'a
    de bornes que celles qui assurent aux autres Membres de la Société la
    jouissance de ces mêmes droits. Ces bornes ne peuvent être déterminées que
    par la Loi. ";

    int index;
    int alpha, blank, punct;

    alpha = blank = punct = 0;

/* Collecte */
    index = 0;
    while(phrase[index])
    {
        if(isalpha(phrase[index]))
            alpha++;
        if(isblank(phrase[index]))
            blank++;
        if(ispunct(phrase[index]))
            punct++;
        index++;
    }

/* Affichage */
    printf("\"%s\"\n", phrase);
    puts("Statistiques :");
    printf("%4d lettres de l'alphabet\n", alpha);
    printf("%4d blancs\n", blank);
    printf("%4d signes de ponctuation\n", punct);

    return(0);
}
```

Remarques

J'avais dit dans les remarques de l'exercice 1301 que le texte de `phrase[]` était réparti sur plusieurs lignes réelles. Si votre opération de copier/coller amène à retrouver plusieurs lignes dans l'éditeur, prenez soin d'enlever les Retour chariot inutiles avant de compiler.

La différence de cette reformulation de ex1301 est au niveau des formateurs `%d` tout à la fin, qui deviennent `%4d`.

J'ai opté pour `%4d` même si l'exemple n'a besoin que de trois positions. Pourquoi ? Pour que le code soit prêt à dénombrer de plus longs textes.

ex1315

```
#include <stdio.h>

int main()
{
    printf("%-9smoi\n", "parle-");
    printf("%-8smoi\n", "parle-");
    printf("%-7smoi\n", "parle-");
    printf("%-6smoi\n", "parle-");
    printf("%-5smoi\n", "parle-");
    printf("%-4smoi\n", "parle-");
    return(0);
}
```

Remarque

Voici l'affichage résultant :

```
parle-   moi
parle-  moi
parle- moi
parle-moi
parle-moi
parle-moi
```

ex1316

```
#include <stdio.h>

int main()
{
    char pres[4][2][11] = {                 // L05
        "George", "Washington",
        "John",   "Adams",
        "Thomas", "Jefferson",
        "James",  "Monroe"
    };
    int boucle;

    for(boucle=0; boucle<4; boucle++)
        printf("%-6s %-10s\n", pres[boucle][0], pres[boucle][1]);  // L14

    return(0);
}
```

Remarques

En ligne 5, nous déclarons bien un tableau à 4 lignes, 2 colonnes de noms avec 10 caractères par nom plus le zéro terminal `\0`.

L'atelier Code::Blocks peut se plaindre à cet endroit en croyant qu'il manque une accolade. Vous l'ignorez.

Nous affichons chaque chaîne à l'aide des variables chaînes `pres[boucle][0]` et `pres[boucle][1]` qui désignent le prénom et le nom de chaque président. Ce ne sont pas des caractères puisque `pres` est un tableau (à trois dimensions).

Les formateurs `%-6s` et `%-10s` demandent d'aligner le texte à gauche avec un espace pour aérer le tout (voyez le formateur de la ligne 14). Le plus long prénom mesure six caractères et le nom le plus long en occupe six (`Washington`).

ex1317

```
#include <stdio.h>

int main()
{
    char i;

    do
    {
        i = getchar();
        putchar(i);
    } while(i != '.');

    putchar('\n');
    return(0);
}
```

Remarques

J'utilise une boucle inversée `do/while` pour garantir au moins un tour de boucle.

La fonction *`getchar()`*, malgré son nom, lit dans un flux. Le traitement se fait un caractère à la fois.

Si vous savez lancer le programme depuis la ligne de commande, vous pouvez demander la redirection en entrée pour lire depuis un fichier. En guise de rappel, si nous supposons que le fichier du programme porte le nom `ex1317`, voici comment procéder :

```
$ ex1317 < ex1317.c
```

CI-dessus, nous demandons de lire le fichier source `ex1317.c`. Voici ce qui s'est affiché en fin d'exécution :

```
#include <stdio.
```

La totalité du contenu a été lu mais l'affichage s'arrête en détectant le premier signe point (condition de sortie). C'est le fonctionnement espéré avec un flux d'entrée.

ex1318

```
#include <stdio.h>

int main()
{
    char primo, secundo;

    printf("Initiale de votre prenom : ");
    primo = getchar();
    printf("Initiale de votre nom de famille : ");
    secundo = getchar();
    printf("Vos initiales sont '%c' et '%c'\n", primo, secundo);
    return(0);
}
```

Remarques

Oui, j'utilise des variables de type `char` quand bien même la fonction `getchar()` renvoie une valeur `int`.

ex1319

```
#include <stdio.h>

char getch(void);

int main()
{
    char primo, secundo;

    printf("Initiale de votre prenom : ");
    primo = getch();
    printf("Initiale de votre nom de famille : ");
    secundo = getch();
    printf("Vos initiales sont '%c' et '%c'\n", primo, secundo);
    return(0);
}

char getch(void)        // Fonction specifique au livre !
{
    char ch;

    ch = getchar();
    while(getchar() != '\n')
        ;
    return(ch);
}
```

Remarques

Cet exemple fonctionne pour l'entrée clavier, mais risque de tomber en boucle si vous lisez depuis un fichier : si le programme atteint la fin du fichier (détection du marqueur `EOF`) avant la lecture du nom de famille, il va attendre jusqu'à la fin du monde. Vous résolvez ce problème en testant la lecture par `getchar()` de `EOF`. Il faut aussi remettre à zéro la condition d'erreur juste après détection de `EOF`. Voici le code corect pour la fonction spécifique `getch()` :

```
char getch(void)
{
    char ch;
    int r;

    ch = getchar();
    do
        r = getchar();
    while( r != EOF && r != '\n' );
    clearerr(stdin);
    return(ch);
}
```

La fonction standard `clearerr()` prend en charge l'erreur causée par la lecture du caractère `EOF` depuis le flux. Rappelons que `stdin` désigne l'entrée standard.

Je remercie le lecteur JSW pour m'avoir donné l'idée de cet exemple.

Chapitre 14

ex1401

```
#include <stdio.h>
```

```
int main()
{
    struct joueur
    {
        char nomj[32];
        int scoremax;
    };
    struct joueur xbox;

    printf("Nom du joueur : ");
    scanf("%s", xbox.nomj);
    printf("Meilleur score : ");
    scanf("%d", &xbox.scoremax);

    printf("Meilleur score de %s : %d\n",      // L17
            xbox.nomj, xbox.scoremax);
    return(0);
}
```

Remarques

La répartition de l'instruction d'affichage `printf()` en ligne 17 ne requiert pas une antibarre de fin de ligne pour poursuivre en ligne suivante. La frappe de Entrée et les espaces ou tabulations en début de ligne suivante sont ignorés par le compilateur. Vous pouvez toujours distribuer une ligne longue ainsi entre les arguments de `printf()`. Nous aurions pu écrire ceci :

```
printf("Meilleur score de %s : %d\n",
   xbox.nomj,
   xbox.scoremax);
```

L'antibarre de saut de ligne n'est obligatoire que dans une chaîne de texte (entre guillemets).

Notez que l'instruction aurait pu tenir sur une ligne, mais cela sert d'exemple.

Voici un exemple de dialogue :

```
Nom du joueur : Billy
Meilleur score : 500
Meilleur score de Billy : 500
```

ex1402

```
#include <stdio.h>

int main()
{
    struct joueur
    {
        char  nomj[32];
        int   scoremax;
        float heuresjeu;
    };
    struct joueur xbox;

    printf("Nom du joueur : ");
    scanf("%s", xbox.nomj);
    printf("Meilleur score : ");
    scanf("%d", &xbox.scoremax);
    printf("Indiquez le nombre d'heures : ");
    scanf("%f", &xbox.heuresjeu);

    printf("Meilleur score de %s : %d\n",
            xbox.nomj, xbox.scoremax);
```

```
    printf("Heures de jeu du joueur %s : %.2f heuresjeu\n",
            xbox.nomj, xbox.heuresjeu);

    return(0);
}
```

Remarques

L'objectif est d'ajouter un quatrième membre de type `float` à la structure de type `joueur`. J'ai choisi d'afficher ce membre avec le formateur `%.2f`. Vous pouvez procéder autrement.

Vous pouvez par exemple combiner déclaration de la structure type et déclaration d'une variable de ce type nommée `xbox` :

```
struct joueur
{
  char nomj[32];
  int scoremax;
  float heuresjeu;
} xbox;
```

Personnellement, je préfère séparer les deux opérations. Je trouve le résultat plus compréhensible.

ex1403

```
#include <stdio.h>

int main()
{
    struct president
    {
        char nomPres[40];
        int  anneeNomi;
    };
    struct president pres1 = {
        "George Washington",
        1789
    };

    printf("Le premier president est %s\n", pres1.nomPres);
    printf("Il a pris ses fonctions en %d\n", pres1.anneeNomi);

    return(0);
}
```

ex1404

```
#include <stdio.h>

int main()
{
    struct president
    {
        char nomPres[40];
        int anneeNomi;
    } pres1 = {
        "George Washington",
        1789
    };

    printf("Le premier president est %s\n", pres1.nomPres);
    printf("Il a pris ses fonctions en %d\n", pres1.anneeNomi);
```

```
    return(0);
}
```

ex1405

```
#include <stdio.h>

int main()
{
    struct president
    {
        char nomPres[40];
        int anneeNomi;
    } pres1 = {                // L09
        "George Washington",
        1789
    };
   struct president pres2 = {
        "John Adams",
        1797
    };

    printf("Le premier president est %s\n", pres1.nomPres);
    printf("Il a pris ses fonctions en %d\n", pres1.anneeNomi);
    printf("Le second president est %s\n", pres2.nomPres);
    printf("Il a pris ses fonctions en %d\n", pres2.anneeNomi);

    return(0);
}
```

Remarques

J'ai indiqué dans le livre que l'initialisation d'une variable à la suite de sa déclaration n'est possible que pour une variable (voir `pres1` en ligne 9). La deuxième variable structure (et les suivantes) doivent être déclarées individuellement.

Voici ce qui est **impossible** :

```
    struct president
    {
        char nomPres[40];
        int  anneeNomi;
    } pres1 = {
        "George Washington",
        1789
    } pres2 = {                /* INTERDIT ! */
        "John Adams",
        1797
    };
```

ex1406

```
#include <stdio.h>

int main()
{
    struct scores
    {
        char nomj[32];
        int  score;
```

```
    };
    struct scores joueurs[4];
    int x;

    for(x=0; x<4; x++)
    {
        printf("Indiquez le joueur %d: ", x+1);
        scanf("%s", joueurs[x].nomj);
        printf("Indiquez son score : ");
        scanf("%d", &joueurs[x].score);
    }

    puts("Infos de joueur");
    printf("#\tNom \tScore\n");                    // L22
    for(x=0; x<4; x++)
    {
        printf("%d\t%s\t%5d\n",                    // L25
            x+1, joueurs[x].nomj, joueurs[x].score);
    }
    return(0);
}
```

Remarques

La ligne 22 devient plus digeste quand vous repérez les méta-caractères de tabulation `\t` pour aligner l'affichage en colonnes. De même en ligne 25.

ex1407

```
#include <stdio.h>

int main()
{
    struct scores
    {
        char nomj[32];
        int  score;
    };
    struct scores joueurs[4];
    struct scores temp;
    int x, a, b;

    for(x=0; x<4; x++)
    {
        printf("Indiquez le joueur %d: ",x+1);
        scanf("%s", joueurs[x].nomj);
        printf("Indiquez son score : ");
        scanf("%d", &joueurs[x].score);
    }

    for(a=0; a<4; a++)                    // L22
        for(b=a+1; b<4; b++)
            if(joueurs[a].score < joueurs[b].score)
            {
                temp = joueurs[a];
                joueurs[a] = joueurs[b];
                joueurs[b] = temp;
            }                             // L29

    puts("Infos de joueur");
    printf("#\tNom\tScore\n");
    for(x=0; x<4; x++)
```

```
    {
        printf("%d\t%s\t%5d\n", x+1, joueurs[x].nomj, joueurs[x].score);
    }
    return(0);
}
```

Remarques

Oui, cet exercice est ardu ! Les projets C ambitieux sont comme lorsque vous voulez manger un éléphant : une bouchée à la fois. Je commence par peupler puis afficher le tableau de structures, ce qui a été fait dans l'exercice 1406. J'ajoute ensuite le code de tri, en m'inspirant des routines du Chapitre 12. Voyez les lignes 22 à 29.

Pour trier, il faut déclarer une variable structure temporaire (ligne 11).

Le tri se fait sur la valeur du membre `score` de la structure `scores`. La comparaison est en ligne 24.

Les structures sont permutées comme des éléments de tableaux normaux (lignes 26 à 28).

Vous n'avez pas besoin de permuter les membres un à un, seulement les structures qui le hébergent. C'est peut-être ce qui vous a le plus étonné à première lecture de la solution.

ex1408

```
#include <stdio.h>
#include <string.h>

int main()
{
    struct date
    {
        int sjour;
        int smois;
        int sannee;
    };
    struct humain
    {
        char hnom[45];
        struct date hdatenaiss;
    };
    struct humain president;

    strcpy(president.hnom,"George Washington");
    president.hdatenaiss.smois = 2;
    president.hdatenaiss.sjour = 22;
    president.hdatenaiss.sannee = 1732;

    printf("Naissance de %s le %d/%d/%d\n",
            president.hnom,
            president.hdatenaiss.sjour,
            president.hdatenaiss.smois,
            president.hdatenaiss.sannee);

    return(0);
}
```

Remarques

N.d.T. : Nous avons permuté les membres de mois et de jour dans la structure de date pour l'adapter à la francophonie.

Pour désigner une sous-structure d'une structure, vous utilisez la notation à point deux fois comme ci-dessus. L'écriture `president.hdatenaiss` désigne un membre de structure, mais c'est aussi une sous-structure. Pour accéder à son membre nommé `sjour`, on écrit donc `president.hdatenaiss.sjour`.

La ligne 19 exploite la fonction standard `strcpy()` pour remplir la variable `president.hnom`. Souvenez-vous que vous ne pouvez pas copier une chaîne avec l'opérateur d'affectation habituel. Il faut la copier avec `strcpy()`. Revoyez le Chapitre 13.

Les structures de type `date` sont souvent imbriquées dans d'autres.

ex1409

```
#include <stdio.h>
#include <string.h>

int main()
{
    struct id
    {
        char hprenom[20];
        char hnomfam[20];
    };
    struct date
    {
        int sjour;
        int smois;
        int sannee;
    };
    struct humain
    {
        struct id hnom;
        struct date hdatenaiss;
    };
    struct humain president;

    strcpy(president.hnom.hprenom,"George");
    strcpy(president.hnom.hnomfam,"Washington");
    president.hdatenaiss.smois = 2;
    president.hdatenaiss.sjour = 22;
    president.hdatenaiss.sannee = 1732;

    printf("Naissance de %s %s le %d/%d/%d\n",
            president.hnom.hprenom,
            president.hnom.hnomfam,
            president.hdatenaiss.sjour,
            president.hdatenaiss.smois,
            president.hdatenaiss.sannee);

    return(0);
}
```

Chapitre 15

ex1501

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    if(argc>1)
        printf("Bienvenue, %s!\n",argv[1]);
    return(0);
}
```

Remarques

Je me souviens avoir écrit un programme dans ce genre à mes débuts. Je l'avais nommé `greet`. J'avais ajouté la commande `greet Dan` dans le fichier .BAT exécuté au démarrage du PC. Tous les matins, j'étais accueilli par le message : "Greetings, Dan!"

ex1502

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Nombre d'arguments fourni = %d.\n", argc);
    return(0);
}
```

ex1503

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Nombre d'arguments fourni = %d.\n", argc);
    printf("Cet argument est %s.\n",argv[0]);
    return(0);
}
```

Remarques

* `argv[0]` contient toujours le premier argument qui est le nom du fichier du programme que vous demandez de lancer.

ex1504

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int x;

    for(x=0; x<argc; x++)
        printf("Arg#%d = %s\n", x+1, argv[x]);   // L08
    return(0);
}
```

Remarques

Dans Code::Blocks, il faut utiliser la commande du menu **Project** nommée **Set Programs' Arguments** pour indiquer les arguments de ligne de commande (programme en mode texte).

Nous utilisons le contenu de `argc` pour contrôler la boucle `for` en ligne 7.

En ligne 8, nous reprenons l'astuce d'incrémentation à la volée par `x+1` pour que les indices affichés commencent à un.

ex1505

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

void sub(void);

int main()
{
    puts("Ce programme stoppe abruptement.");
```

```
    sub();
    puts("Est-ce l'intention initiale ?");
    return(0);
}

void sub(void)
{
    puts("C'est normal.");
    exit(0);
}
```

Remarques

La fonction `exit()` sert à sortir sur le champ, pas comme `return`. L'exécution s'arrête sur la ligne de l'appel à `exit()`.

Même si vous sortez avec `exit()`, je vous conseille de prévoir un `return` dans la fonction `main()`. Au minimum, cela vous évitera le message de compilation inutile d'atteinte de la fin d'une fonction non void ("control reaches end of non-void function").

ex1506 - solution Windows

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

int main()
{
    printf("Validez par Entree pour vider l'affichage :");
    getchar();
    system("cls");
    puts("C'est mieux.");
    return(0);
}
```

Remarques

Nous ne faisons rien de la valeur lue par `getchar()` qui ne sert qu'à marquer une pause.

ex1506 - solution MacOS/Unix/Linux

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

int main()
{
    printf("Validez par Entree pour vider l'affichage :");
    getchar();
    system("clear");
    puts("C'est mieux.");
    return(0);
}
```

Remarques

Voyez la remarque de la version Windows.

Chapitre 16

ex1601

```
#include <stdio.h>

int main()
```

```
{
    int a,b;
    float c;

    printf("Premier entier : ");
    scanf("%d", &a);
    printf("Second entier : ");
    scanf("%d", &b);
    c = a/b;
    printf("%d/%d = %.2f\n", a, b, c);
    return(0);
}
```

ex1602

```
#include <stdio.h>

int main()
{
    int a, b;
    float c;

    printf("Premier entier : ");
    scanf("%d", &a);
    printf("Second entier : ");
    scanf("%d", &b);
    c = (float)a / (float)b;
    printf("%d/%d = %.2f\n", a, b, c);
    return(0);
}
```

Remarques

En ligne 12, la solution intermédiaire `(float)(a/b)` ne conviendrait pas parce que les deux valeurs sont des entiers et le résultat aussi : nous le transtypons en flottant après la perte de précision, mais c'est trop tard. L'entier arrondi 1 (par exemple) devient le flottant 1.0 (ce qui nous fait une belle jambe).

Nous avons déjà fait un transtypage dans le Chapitre 11 avec la fonction `srand()` :

```
srand((unsigned)time(NULL));
```

Nous transtypons vers le type non signé `(unsigned)`. En effet, la fonction `time()` renvoie une valeur signée, mais `srand()` veut une `unsigned`. Cette précaution nous épargne un avertissement et des surprises à l'affichage dans le cas où nous alimenterions `srand()` avec une valeur signée.

ex1603

```
#include <stdio.h>

typedef int tentier;

tentier main()
{
    tentier a = 2;

    printf("Tout le monde sait que ");
    printf("%d + %d = %d\n", a, a, a+a);
    return(0);
}
```

Remarques

Vous trouverez de bons exemples d'utilisation de `typedef` en parcourant les fichiers d'en-tête fournis avec le compilateur, notamment pour définir des types de variables dans les prototypes de fonctions. Les définitions de types permettent plus de cohérence, sans entraîner trop de rigidité. En effet, vous pouvez toujours redéfinir localement un `typedef`, comme une constante, dans un fichier d'en-tête spécifique. Inutile de traquer les autres occurrences de ce type dans les autres fichiers.

ex1604

```
#include <stdio.h>
#include <string.h>

int main()
{
    typedef struct id
    {
        char hprenom[20];          // primo dans Listing 16.4
        char hnomfam[20];          // nomfam dans Listing 16.4
    } personne;

    typedef struct date
    {
        int sjour;
        int smois;
        int sannee;
    } calendrier;

    struct humain
    {
        personne hnom;
        calendrier hdatenaiss;
    };
    struct humain president;        // L24

    strcpy(president.hnom.hprenom,"George");
    strcpy(president.hnom.hnomfam,"Washington");
    president.hdatenaiss.sjour = 22;
    president.hdatenaiss.smois = 2;
    president.hdatenaiss.sannee = 1732;

    printf("Naissance de %s %s le %d/%d/%d\n",\
            president.hnom.hprenom,
            president.hnom.hnomfam,
            president.hdatenaiss.sjour,
            president.hdatenaiss.smois,
            president.hdatenaiss.sannee);

    return(0);
}
```

Remarques

Le mot clé *typedef* ne modifie pas la façon de faire référence aux variables ; il ne concerne que le type des variables.

N'oubliez pas que les mots `personne` et `calendrier` dans ce code sont des noms de types locaux, pas des noms de variables ! La seule variable structure déclarée ici est `president` en ligne 24.

ex1605

```
#include <stdio.h>
```

```
void proc(void);

int main()
{
    puts("Premier appel");
    proc();
    puts("Second appel");
    proc();
    return(0);
}

void proc(void)
{
    int a;

    printf("La valeur de la variable a est %d\n", a);
    printf("Indiquez une autre valeur : ");
    scanf("%d", &a);
}
```

Remarques

Oui, dans ce code, je fais la faute grave consistant à utiliser une variable avant de l'avoir initialisée (dans `proc()`), mais c'est voulu et cette variable ne contient pas un paramètre vital d'un pilote automatique d'avion de ligne. Le but est de montrer que la valeur de la variable saisie est perdue en sortie de fonction (ligne 20).

Le compilateur vous préviendra sans doute que la variable `a` n'est pas initialisée.

Par un hasard incroyable, mais vous devez me croire, lors de mes essais, la valeur trouvée par le programme à l'emplacement de la variable était zéro ! Ce n'est qu'une des millions d'autres valeurs que peut représenter la case mémoire d'un entier. Chez vous, vous verrez par exemple 1964477653. C'est normal.

ex1606

```
#include <stdio.h>

void proc(void);

int main()
{
    puts("Premier appel");
    proc();
    puts("Second appel");
    proc();
    return(0);
}

void proc(void)
{
    static int a;                           // Retouche

    printf("La valeur de la variable a est %d\n", a);
    printf("Indiquez une autre valeur : ");
    scanf("%d", &a);
}
```

Remarque

Une variable déclarée `static` fait l'objet d'une initialisation à zéro par le compilateur. Vous le confirmez à l'affichage et cela explique pour quoi le compilateur ne se plaint plus par un avertissement.

ex1607

```
#include <stdio.h>

void moitier(void);
void doubler(void);

int   age;
float toise;

int main()
{
    printf("Quel est votre age : ");
    scanf("%d", &age);
    printf("Et votre taille : ");
    scanf("%f", &toise);
    printf("Vous avez %d ans et mesures %.1f.\n", age, toise);
    moitier();
    doubler();
    printf("Vous n'avez pas %d ans et ne mesurez pas %.1f.\n", age, toise);
    return(0);
}

void moitier(void)
{
    float a,h;

    a=(float)age/2.0;
    printf("La moitie de votre age est %.1f.\n", a);
    h=toise/2.0;
    printf("La moitie de votre taille est %.1f.\n", h);

}

void doubler(void)
{
    age*=2;
    printf("Le double de votre age est %d.\n", age);
    toise*=2;
    printf("Le double de votre taill est %.1f\n", toise);
}
```

ex1608

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

#define TAILLE 5

struct bot {
    int xpos;
    int ypos;
};

struct bot initialiser(struct bot b);

int main()
{
    struct bot robots[TAILLE];
```

```
    int x;

    srandom((unsigned)time(NULL));           // L19

    for(x=0; x<TAILLE; x++)
    {
      robots[x] = initialiser(robots[x]);
      printf("Robot %d: Coord.: %d,%d\n", x+1, robots[x].xpos, robots[x].ypos);
    }
    return(0);
}

struct bot initialiser(struct bot b)
{
    int x,y;

    x = random();
    y = random();
    x%=20;
    y%=20;
    b.xpos = x;
    b.ypos = y;
    return(b);
}
```

Remarques

Notez au passage le transtypage en ligne 19.

En déclarant la structure de façon globale, nous pouvez ensuite déclarer des fonctions avec ce type, ce qui est le cas de `initialiser()` en ligne 30. C'est d'ailleurs le seul moyen de coder une fonction devant renvoyer une structure.

La structure est une solution pour faire renvoyer plusieurs valeurs à la fois depuis une fonction.

Chapitre 17

ex1701

```
#include <stdio.h>

char *binbin(int n);

int main()
{
    int input;

    printf("Indiquez une valeur entre 0 et 255 : ");
    scanf("%d", &input);
    printf("%d vaut en binaire 8 bits %s\n", input, binbin(input));
    return(0);
}

char *binbin(int n)
{
    static char bin[9];
    int x;

    for(x=0;x<8;x++)
    {
```

```
        bin[x] = n & 0x80 ? '1' : '0';
        n <<= 1;
    }
    bin[x] = '\0';
    return(bin);
}
```

ex1702

```
#include <stdio.h>

char *binbin(int n);

int main()
{
    int input;

    printf("Entrez une valeur entre 0 et 65535 : ");
    scanf("%d", &input);
    printf("%d vaut en binaire 16 bits %s\n", input, binbin(input));
    return(0);
}

char *binbin(int n)
{
    static char bin[17];
    int x;

    for(x=0;x<16;x++)
    {
        bin[x] = n & 0x8000 ? '1' : '0';
        n <<= 1;
    }
    bin[x] = '\0';
    return(bin);
}
```

ex1703

```
#include <stdio.h>
#define SET 32

char *binbin(int n);

int main()
{
    int bor, resultat;

    printf("Indiquez une valeur entre 0 et 255 : ");
    scanf("%d", &bor);
    resultat = bor | SET;

    printf("\t%s\t%d\n",  binbin(bor), bor);
    printf("|\t%s\t%d\n", binbin(SET), SET);
    printf("=\t%s\t%d\n", binbin(resultat), resultat);
    return(0);
}

char *binbin(int n)
{
    static char bin[9];
    int x;
```

```
    for(x=0; x<8; x++)
    {
        bin[x] = n & 0x80 ? '1' : '0';
        n <<= 1;
    }
    bin[x] = '\0';
    return(bin);
}
```

ex1704

```
#include <stdio.h>

int main()
{
    char input[64];
    int ch;
    int x = 0;

    printf("Saisissez en MAJUSCULES : ");
    fgets(input, 63, stdin);

    while(input[x] != '\n')
    {
        ch = input[x] | 32;
        putchar(ch);
        x++;
    }
    putchar('\n');

    return(0);
}
```

Remarques

Vous saurez tout : je me suis trompé au départ en déclarant la variable `ch` de type *char* alors que tout le monde sait que la fonction `putchar()` attend en entrée une valeur `int`. La compilation réussit néanmoins, mais vous pourriez avoir des surprises pour les valeurs supérieures à 255.

ex1705

```
#include <stdio.h>
#define SET 223

char *binbin(int n);

int main()
{
    int bor,resultat;

    printf("Indiquez une valeur entre 0 et 255 : ");
    scanf("%d", &bor);
    resultat = bor & SET;

    printf("\t%s\t%d\n",  binbin(bor), bor);
    printf("&\t%s\t%d\n", binbin(SET), SET);
    printf("=\t%s\t%d\n", binbin(resultat), resultat);
    return(0);
}

char *binbin(int n)
```

```
{
    static char bin[9];
    int x;

    for(x=0;x<8;x++)
    {
        bin[x] = n & 0x80 ? '1' : '0';
        n <<= 1;
    }
    bin[x] = '\0';
    return(bin);
}
```

ex1706

```
#include <stdio.h>

int main()
{
    char input[64];
    char ch;
    int x = 0;

    printf("Saisissez du texte : ");
    fgets(input, 63, stdin);

    while(input[x] != '\n')
    {
        ch = input[x] & 223;
        putchar(ch);
        x++;
    }
    putchar('\n');

    return(0);
}
```

Remarques

L'affichage varie selon le système. Sur PC sous Windows, on voit ceci :

```
Saisissez du texte : un petit test
UN PETIT TEST
```

Sous MacOS :

```
Saisissez du texte : un petit test
UNPETITTEST
```

C'est lié au jeu de caractères en vigueur dans le terminal. Sur PC, le code ASCII 0 peut apparaître comme un espace, mais disparaît sous MacOS. De même si vous saisissez des caractères non affichables :

```
Saisissez du texte : 8*8=64
↑
↑⇔¶
```

Les signes bizarres sont produits par le masque binaire appliqué au code ASCII.

L'espace correspond à la valeur ASCII 32. Si vous indiquez cette valeur dans ex1705, l'affichage est celui-ci :

```
Indiquez une valeur entre 0 et 255 : 32
   00100000    32
&  11011111    223
=  00000000    0
```

Le masque ET à 223 sur le code de l'espace ramène le code à zéro.

ex1707

```
#include <stdio.h>
#include <ctype.h>

int main()
{
    char input[64];
    char ch;
    int x = 0;

    printf("Saisissez du texte : ");
    fgets(input, 63, stdin);

    while(input[x] != '\n')
    {
        if(isalpha(input[x]))           // L15
            ch = input[x] & 223;
        else
            ch = input[x];
        putchar(ch);
        x++;
    }
    putchar('\n');

    return(0);
}
```

Remarque

J'ai profité de la fonction de la famille CTYPE nommée `isalpha()` en ligne 15. Elle renvoie TRUE pour une lettre de l'alphabet, et FALSE sinon.

ex1708

```
#include <stdio.h>

char *binbin(int n);

int main()
{
    int a,x,r;

    a = 73;
    x = 170;

    printf("  %s %3d\n", binbin(a),a);
    printf("^ %s %3d\n", binbin(x),x);
    r = a ^ x;
    printf("= %s %3d\n", binbin(r),r);
    return(0);
}

char *binbin(int n)
{
    static char bin[9];
    int x;

    for(x=0;x<8;x++)
    {
```

```
        bin[x] = n & 0x80 ? '1' : '0';
        n <<= 1;
    }
    bin[x] = '\0';
    return(bin);
}
```

Remarques

Exemple d'affichage résultant :

```
    01001001 73
  ^ 10101010 170
  = 11100011 227
```

Quand le bit de même rang des deux valeurs est différent, le XOR force la sortie à 1. S'ils sont identiques (deux 1 ou deux 0), le XOR force à 0.

ex1709

```
#include <stdio.h>

char *binbin(int n);

int main()
{
    int a,x,r;

    a = 73;
    x = 170;

    printf("  %s %3d\n", binbin(a),a);
    printf("^ %s %3d\n", binbin(x),x);
    r = a ^ x;
    printf("= %s %3d\n", binbin(r),r);
    printf("^ %s %3d\n", binbin(x),x);
    a = r ^ x;                              // L17
    printf("= %s %3d\n", binbin(a),a);
    return(0);
}

char *binbin(int n)
{
    static char bin[9];
    int x;

    for(x=0;x<8;x++)
    {
        bin[x] = n & 0x80 ? '1' : '0';
        n <<= 1;
    }
    bin[x] = '\0';
    return(bin);
}
```

Remarques

Je réutilise en ligne 17 la variable `a` pour y stocker le résultat du XOR (OUEX). C'est dans la logique du XOR de retrouver dans `a` la valeur de départ 73.

L'opérateur XOR permet de concevoir vite fait, bien fait un codeur/décodeur de mot de passe. Sachez qu'il est facile à craquer, mais l'exemple illustre les capacités de l'opérateur `^`. Voici un exemple complet que je vous propose de tester et d'étudier :

```c
#include <stdio.h>

char getch(void);

int main()
{
        char motsec[20];
        char crypteur[20];
        int code, i;

        printf("Saisissez un mot secret : ");
        scanf("%s", motsec);
        printf("Indiquez un code de cryptage (0 a 255) : ");
        scanf("%d", &code);

        i = 0;
        while(motsec[i])
        {
                crypteur[i] = motsec[i] ^ code;
                i++;
        }
        printf("Voici le mot illisible : '%s'\n", crypteur);

        printf("Frappez Entree pour decrypter...");
        getch();

        i = 0;
        while(motsec[i])
        {
                motsec[i] = crypteur[i] ^ code;
                i++;
        }
        printf("Et revoici le mot secret lisible : '%s'\n", motsec);

        return(0);
}

char getch(void)
{
    char ch;
    int r;

    ch = getchar();
    do
        r = getchar();
    while( r != EOF && r != '\n' );
    clearerr(stdin);
    return(ch);
}
```

ex1710

```c
#include <stdio.h>

char *binbin(int n);

int main()
{
    int decalbin, x;
```

```
    printf("Indiquez une valeur entre 0 et 255 : ");
    scanf("%d", &decalbin);

    for(x=0;x<8;x++)
    {
        printf("%s\n", binbin(decalbin));
        decalbin = decalbin << 1;
    }

    return(0);
}

char *binbin(int n)
{
    static char bin[9];
    int x;

    for(x=0; x<8; x++)
    {
        bin[x] = n & 0x80 ? '1' : '0';
        n <<= 1;
    }
    bin[x] = '\0';
    return(bin);
}
```

Remarque

Voici l'affichage pour la saisie de la valeur 1 :

```
Indiquez une valeur entre 0 et 255 : 1
00000001
00000010
00000100
00001000
00010000
00100000
01000000
10000000
```

Vous voyez le bit à 1 progresser vers la gauche du masque d'octet.

N.d.T. : Il progresse en direction du bit de poids fort (MSB, *Most Significant Bit*).

ex1711

```
#include <stdio.h>

char *binbin(int n);

int main()
{
    int decalbin, x;

    printf("Indiquez une valeur entre 0 et 255 : ");
    scanf("%d", &decalbin);

    for(x=0;x<8;x++)
    {
        printf("%s %d\n", binbin(decalbin), decalbin);
        decalbin = decalbin << 1;
    }
```

```
    return(0);
}

char *binbin(int n)
{
    static char bin[17];
    int x;

    for(x=0; x<16; x++)
    {
        bin[x] = n & 0x8000 ? '1' : '0';
        n <<= 1;
    }
    bin[x] = '\0';
    return(bin);
}
```

ex1712

```
#include <stdio.h>

char *binbin(int n);

int main()
{
    int decalbin, x;

    printf("Indiquez une valeur entre 0 et 255 : ");
    scanf("%d", &decalbin);

    for(x=0;x<8;x++)
    {
        printf("%s %d\n", binbin(decalbin), decalbin);
        decalbin = decalbin >> 1;
    }

    return(0);
}

char *binbin(int n)
{
    static char bin[17];
    int x;

    for(x=0; x<16; x++)
    {
        bin[x] = n & 0x8000 ? '1' : '0';
        n <<= 1;
    }
    bin[x] = '\0';
    return(bin);
}
```

ex1713

```
#include <stdio.h>

char *binbin(int n);
```

```
int main()
{
    int b, x;

    b = 21;

    for(x=0;x<8;x++)
    {
        printf("%s 0x%04X %4d\n", binbin(b),b,b);
        b<<=1;
    }

    return(0);
}

char *binbin(int n)
{
    static char bin[17];
    int x;

    for(x=0; x<16; x++)
    {
        bin[x] = n & 0x8000 ? '1' : '0';
        n <<= 1;
    }
    bin[x] = '\0';
    return(bin);
}
```

Remarques

Exemple d'affichage :

```
0000000000010101 0x0015 21
0000000000101010 0x002A 42
0000000001010100 0x0054 84
0000000010101000 0x00A8 168
0000000101010000 0x0150 336
0000001010100000 0x02A0 672
0000010101000000 0x0540 1344
0000101010000000 0x0A80 2688
```

Les valeurs hexadécimales de chaque quartet de bits sont malaisées à repérer, mais munissez-vous du Tableau 17.4 du livre pour y arriver.

Voici le même affichage après mise en gras d'un groupe sur deux :

```
0000000000010101 0x0015 21
0000000000101010 0x002A 42
0000000001010100 0x0054 84
0000000010101000 0x00A8 168
0000000101010000 0x0150 336
0000001010100000 0x02A0 672
0000010101000000 0x0540 1344
0000101010000000 0x0A80 2688
```

ex1714

```
#include <stdio.h>

char *binbin(int n);
```

```
int main()
{
    int b,x;

    b = 0x11;

    for(x=0; x<8; x++)
    {
        printf("%s 0x%04X %4d\n", binbin(b),b,b);
        b<<=1;
    }

    return(0);
}

char *binbin(int n)
{
    static char bin[17];
    int x;
     int x;

    for(x=0; x<16; x++)
    {
        bin[x] = n & 0x8000 ? '1' : '0';
        n <<= 1;
    }
    bin[x] = '\0';
    return(bin);
}
```

Suite en Partie IV, Chapitre 18