

Les bases : exercices résolus en C

Corrigé

Objectifs

- Raffiner des problèmes simples ;
- Écrire quelques algorithmes simples ;
- Savoir utiliser les types de base ;
- Savoir utiliser les instructions « élémentaires » : d’entrée/sortie, affichage, bloc...
- Manipuler les conditionnelles ;
- Manipuler les répétitions.

Exercice 1 : Conversion pouce/centimètre	1
Exercice 2 : Quelques statistiques	7
Exercice 3 : Division entière	13
Exercice 4 : Nombres premiers	16
Exercice 5 : Nombres parfaits	22

Exercice 1 : Conversion pouce/centimètre

Écrire un programme qui réalise la conversion pouce/centimètre d’une longueur saisie au clavier. Une longueur sera saisie comme un nombre réel suivi d’un caractère précisant l’unité. Les unités possibles sont le pouce (p), le centimètre (c) ou le mètre (m). Le programme affichera la longueur exprimée en pouce et en centimètre.

Voici des exemples d’exécution du programme :

```
Entrez une longueur : 1p
1 p = 2.54 cm
```

```
Entrez une longueur : 2m
78.7402 p = 200 cm
```

```
Entrez une longueur : 2km
0 p = 0 cm
```

Modifier le programme pour permettre la saisie de l’unité aussi bien en minuscules qu’en majuscules.

Solution :

R0 : Afficher une longueur saisie au clavier en pouces et en centimètres

Nous reprenons les jeux de test proposés dans le sujet. Nous ajoutons un jeu de test qui consiste à saisir une unité en centimètres. Par exemple, 5.08 cm qui doit donner 1 p = 5.08 p.

Lorsque l'on regarde les résultats d'exécution donnés dans le sujet (et qui servent donc de spécification pour le programme à écrire), on constate que, quelle que soit l'unité utilisée pour saisir la longueur, il faut afficher d'abord la longueur en pouces, puis son équivalent en centimètres. En conséquence, on en déduit qu'il faut calculer les deux longueurs (une dans chaque unité). Ainsi, après saisi de la longueur de l'utilisateur, nous allons la convertir dans les deux unités puis afficher la ligne d'équivalence.

```

1  R1 : Raffinage De « Afficher une longueur saisie au clavier... »
2  | Saisir la longueur          valeur: out Réel ; unité: out Caractère
3  | Calculer la longueur en pouces et en centimètres
4  |                             valeur, unité : in
5  |                             lg_p  : out Réel -- longueur en pouces
6  |                             lg_cm : out Réel -- longueur en centimètres
7  | Afficher le résultat      lg_p, lg_cm : in

```

La seule étape délicate est la deuxième. Calculer les longueurs en pouces et en centimètres dépend de l'unité saisie par l'utilisateur. Il s'agit donc d'utiliser une conditionnelle et plus précisément de faire un traitement par cas. Puisque l'unité est un caractère, donc un type scalaire, nous pouvons utiliser un **Selon**.

Remarque : Pour traiter le cas des majuscules, il suffit d'ajouter le cas majuscule à côté du cas minuscule correspondant.

On aurait également pu convertir la longueur saisie en minuscule avant de calculer les longueurs en pouces et centimètres.

Remarque : On aurait également pu utiliser des **Si** et **SiNonSi**. Cependant, la structure du programme est plus claire en utilisant un **Selon**. Ceci est d'autant plus vrai si l'on traite également les majuscules.

Voici le raffinement correspondant.

```

1  R2 : Raffinage De « Calculer la longueur en pouces et en centimètres »
2  | Selon unité Dans
3  |   'p', 'P':          { la longueur a été saisie en pouces }
4  |   lg_p <- valeur
5  |   lg_cm <- lg_p * UN_POUCE
6  |
7  |   'c', 'C':          { la longueur a été saisie en centimètres }
8  |   lg_cm <- valeur
9  |   lg_p <- lg_cm / UN_POUCE
10 |
11 |   'm', 'M':          { la longueur a été saisie en mètres }
12 |   lg_cm <- valeur * 100
13 |   lg_p <- lg_cm / UN_POUCE
14 |
15 | Sinon                { Unité non reconnue }
16 |   lg_p <- 0
17 |   lg_cm <- 0
18 | FinSelon

```

Notons que nous avons utilisé une constante symbolique (UN_POUCE) plutôt qu'une constante littérale (2,54). L'intérêt est d'avoir un programme plus lisible et d'éviter la redondance (si on

se trompe sur la valeur d'un pouce en centimètres il suffit de faire un seul changement pour corriger le programme dans le cas de la constante symbolique contre trois sinon).

```

1  Algorithme pouce2cm
2
3      -- Afficher une longueur saisie au clavier en pouces et en centimètres.
4
5  Constantes
6      UN_POUCE = 2.54      -- valeur en centimètres d'un pouce
7  Variables
8      valeur: Réel        -- valeur de la longueur lue au clavier
9      unité: Caractère    -- unité de la longueur lue au clavier
10     lg_cm: Réel         -- longueur exprimée en centimètres
11     lg_p: Réel          -- longueur exprimée en pouces
12
13 Début
14     -- saisir la longueur (valeur + unité)
15     Écrire ("Entrer_une_longueur_(valeur+_unité)_:_")
16     Lire (valeur)          -- saisir la valeur
17     Lire (unité)         -- saisir l'unité
18
19     -- calculer la longueur en pouces et en centimètres
20     Selon unité Dans
21         'p', 'P':          { la longueur a été saisie en pouces }
22             lg_p <- valeur
23             lg_cm <- lg_p * UN_POUCE
24
25         'c', 'C':          { la longueur a été saisie en centimètres }
26             lg_cm <- valeur
27             lg_p <- lg_cm / UN_POUCE
28
29         'm', 'M':          { la longueur a été saisie en mètres }
30             lg_cm <- valeur * 100
31             lg_p <- lg_cm / UN_POUCE
32
33     Sinon                  { Unité non reconnue }
34         lg_p <- 0
35         lg_cm <- 0
36     FinSelon
37
38     -- afficher le résultat
39     ÉcrireLn (lg_p, "_p_", lg_cm, "_cm")
40 Fin

1  /*****
2  *  Auteur   : Xavier CRÉGUT
3  *  Version : Revision : 1.6
4  *  Objectif : Conversion pouces/centimètres
5  *
6  *  Remarque : Le sujet n'est pas entièrement respecté car la partie
7  *  décimale est affichée même si elle est nulle !
8  *****/

```

```
9
10 #include <stdio.h>
11 #include <stdlib.h>
12
13 const double UN_POUCE = 2.54; /* Valeur de 1 pouce en centimètres */
14
15 int main()
16 {
17     double valeur; /* valeur lue au clavier */
18     char unite; /* unité lue au clavier */
19     /* valeur et unité forment la longueur */
20     double lg_p; /* longueur lue au clavier convertie en pouces */
21     double lg_cm; /* longueur lue au clavier convertie en centimètres */
22
23     /* Saisir la longueur */
24     printf("Entrez une longueur : ");
25     scanf("%lf%c", &valeur, &unite);
26     /* Remarque : l'espace devant %c est important : il permet de
27      * sauter les caractères blancs. C'est ce qui autorise par exemple
28      * à saisir la longueur sous la forme « 12 p ».
29      */
30
31     /* Calculer la longueur en pouces et en centimètres */
32     switch (unite) {
33         case 'p':
34         case 'P': /* longueur saisie en pouces */
35             lg_p = valeur;
36             lg_cm = valeur * UN_POUCE;
37             break;
38
39         case 'm':
40         case 'M': /* longueur saisie en mètres */
41             lg_p = 100 * valeur / UN_POUCE;
42             lg_cm = 100 * valeur;
43             break;
44
45         case 'c':
46         case 'C': /* longueur saisie en centimètres */
47             lg_p = valeur / UN_POUCE;
48             lg_cm = valeur;
49             break;
50
51         default:
52             lg_p = lg_cm = 0;
53     }
54
55     /* Afficher les résultats */
56     printf("%.1f_p = %.1f_cm\n", lg_p, lg_cm);
57
58     return EXIT_SUCCESS;
59 }
```

Voici une version qui tient compte de la particularité du **switch**. Attention, cet exemple n'est donné que pour que vous compreniez bien le fonctionnement du **switch** mais il faut éviter d'avoir un cas qui ne se termine pas par un **break**. En effet, l'habitude étant de mettre un **break**, le lecteur ne verra pas nécessairement son absence. D'autre part, ceci signifie que pour le cas « 'c' », il y a deux points d'entrée, le premier évident si l'unité vaut 'c' le second, moins facile à voir, si l'unité vaut 'm'. Si l'on change les instructions du cas 'c', il faudra bien vérifier qu'elles restent cohérentes pour les deux points d'entrée !

```

1  /*****
2  *  Auteur   : Xavier CRÉGUT
3  *  Version : 1.1
4  *  Objectif : Conversion pouces/centimètres
5  *
6  *  Remarque : Le sujet n'est pas entièrement respecté car la partie
7  *  décimale est affichée même si elle est nulle !
8  *****/
9
10 #include <stdio.h>
11 #include <stdlib.h>
12
13 #define UN_POUCE      2.54    /* Valeur de 1 pouce en centimètres */
14
15 int main()
16 {
17     double valeur;          /* valeur lue au clavier */
18     char unite;             /* unité lue au clavier */
19     /* valeur et unité forment la longueur */
20     double p;              /* longueur lue au clavier convertie en pouces */
21     double cm;             /* longueur lue au clavier convertie en centimètres */
22
23     /* Saisir la longueur */
24     printf("Entrez une longueur : ");
25     scanf("%lf%c", &valeur, &unite);
26     /* Remarque : l'espace devant %c est important : il permet de
27      * sauter les caractères blancs. C'est ce qui autorise par exemple
28      * à saisir la longueur sous la forme « 12 p ».
29      */
30
31     /* Calculer la longueur en pouces et en centimètres */
32     switch (unite) {
33         case 'p':
34             case 'P':          /* longueur saisie en pouces */
35                 p = valeur;
36                 cm = valeur * UN_POUCE;
37                 break;
38
39         case 'm':
40             case 'M':          /* longueur saisie en mètres */
41                 valeur = valeur * 100;
42                 /* on continue en exécutant les instructions du cas 'c' */
43                 /* Attention : ceci est possible, correct mais dangereux ! */

```

```
44     case 'c':
45     case 'C':           /* longueur saisie en centimètres */
46         p = valeur / UN_POUCE;
47         cm = valeur;
48         break;
49
50     default:
51         p = cm = 0;
52 }
53
54 /* Afficher les résultats */
55 printf("%1.4f_p_=%1.4f_cm\n", p, cm);
56
57 return EXIT_SUCCESS;
58 }
```

Exercice 2 : Quelques statistiques

L'objectif de cet exercice est de calculer quelques statistiques sur une série de valeurs réelles. Cette série est lue au clavier. On considère qu'elle se termine par la valeur nulle (0) qui ne fait pas partie de la série.

2.1 Écrire un programme qui détermine la moyenne des valeurs de la série.

Solution : Pour calculer la moyenne, j'ai besoin de la somme des valeurs et du nombre de valeurs. On peut donc en déduire les deux variables d'accumulation suivantes :

```

1 Variables
2     nb: Entier -- le nombre de valeurs lues de la série
3     somme: Réel -- la somme des valeurs lues de la série

```

Les premiers niveaux de raffinement peuvent être présentés ainsi :

```

1 R0 : Afficher la moyenne d'une série de valeurs réelles
2
3 R1 : Raffinage De « Afficher la moyenne d'une série de valeurs réelles »
4     | Déterminer le nb d'éléments dans la série et leur somme
5     | Afficher le résultat
6
7 R2 : Raffinage De « Déterminer le nb d'éléments dans la série et leur somme »
8     | Initialiser : somme <- 0 et nb <- 0
9     | Saisir le premier réel x
10    | TantQue x <> 0 Faire
11    |     | Mettre à jour les variables somme et nb en fonction de x
12    |     | Saisir le réel suivant x
13    | FinTQ
14
15 R2 : Raffinage De « Afficher le résultat »
16    | Si nb > 0 Alors
17    |     | Afficher la moyenne = somme / nb
18    | Sinon
19    |     | Signaler moyenne impossible
20    | FinSi

```

Remarque : On pouvait pas utiliser un **Pour** car on ne connaît pas à priori le nombre de valeurs dans la série. Ceci dépend de l'utilisateur. C'est lui qui décide de saisir 0. On doit donc utiliser soit un **TantQue**, soit un **Répéter**.

Nous avons utilisé un **TantQue** car nous avons considéré « traiter une valeur » qui peut ne pas être exécutée (cas où l'utilisateur saisit tout de suite 0 pour indiquer que la série est vide).

Nous aurions également pu utiliser un **Répéter** en nous intéressant à la saisie d'un réel. L'utilisateur doit nécessairement saisir au moins un réel (une valeur de la série ou zéro). Notons que dans ce cas, il faut ensuite faire un test pour savoir si le réel saisi fait partie de la série et, dans l'affirmative, le comptabiliser.

Voici le raffinement correspondant.

```

1 Initialiser les variables statistiques avec x
2 Répéter
3     | Saisir un réel                               x: out
4     | Si x <> 0 Alors { x est une valeur de la série }
5     |     | Mettre à jour les variables statistiques

```

```

6   | FinSi
7   Jusqu'à x = 0

```

On remarque que dans le cas du **TantQue** on duplique une instruction (la saisie d'un réel) et dans le cas du **Répéter** on duplique un condition.

L'algorithme peut ensuite être le suivant :

```

1  Algorithme statistiques_simples_moyenne
2
3      -- Afficher la moyenne d'une série de valeurs réelles lues au clavier.
4      -- La série est terminée par zéro qui n'appartient pas à la série.
5
6  Variables
7      x: Réel      -- un réel lu au clavier
8      nb: Entier  -- le nombre de valeurs dans la série
9      somme: Réel -- la somme des valeurs lues de la série
10
11 Début
12     -- Déterminer le nb d'éléments dans la série et leur somme
13     -- Initialiser les variables
14     somme <- 0 -- pas encore d'éléments lus
15     nb <- 0
16     -- Saisir le premier réel
17     Lire(x)
18     -- Traiter les éléments de la série
19     TantQue x <> 0 Faire
20         -- Mettre à jour les variables somme et nb en fonction de x
21         somme <- somme + x
22         nb <- nb + 1
23
24         -- Saisir le réel suivant
25         Lire(x)
26     FinTQ
27
28     -- Afficher le résultat
29     Si nb > 0 Alors                                { La moyenne a un sens }
30         -- Afficher la moyenne
31         ÉcrireLn("Moyenne_=", somme / nb)
32     Sinon                                           { La moyenne n'a PAS de sens }
33         -- Signaler moyenne impossible
34         ÉcrireLn("La_série_est_vide._La_moyenne_n'existe_donc_pas!")
35     FinSi
36 Fin.

1  /*****
2  *  Auteur   :  Xavier Crégut <cregut@enseeiht.fr>
3  *  Version :  1.2
4  *
5  *  Objectif :
6  *           Afficher la moyenne d'une série de valeurs réelles lues au clavier.
7  *
8  *****/

```



```
9
10 #include <stdio.h>
11 #include <stdlib.h>
12
13 int main()
14 {
15     double x;           /* un réel lu au clavier */
16     int nb;             /* le nombre de valeurs lues de la série */
17     double somme;       /* la somme des valeurs lues de la série */
18
19     /* afficher la consigne */
20     printf("Donnez une série d'entiers qui se termine par 0.\n");
21
22     /* déterminer le nb d'éléments dans la série et leur somme */
23     /* initialiser les variables */
24     somme = 0;          /* pas encore d'éléments lus */
25     nb = 0;
26
27     /* lire le premier réel */
28     scanf("%lf", &x);
29
30     /* traiter les éléments de la série */
31     while (x != 0) {
32         /* mettre à jour les variables somme et nb en fonction de x */
33         somme = somme + x;
34         nb++;
35
36         /* lire le réel suivant */
37         scanf("%lf", &x);
38     }
39
40     /* afficher le résultat */
41     if (nb > 0) {       /* La moyenne a un sens */
42         /* afficher la moyenne */
43         printf("Moyenne = %f\n", somme / nb);
44     }
45     else {              /* La moyenne n'a PAS de sens */
46         /* signaler moyenne impossible */
47         printf("La série est vide. La moyenne n'existe donc pas!\n");
48     }
49
50     return EXIT_SUCCESS;
51 }
52
53 /*
54 1 2 3 0    -->    2
55 2 -2 0     -->    0
56 -4 -2 0    -->   -3
57 13 0       -->   13
58 0          -->   Non défini
59 */
```

2.2 En plus de la moyenne, on veut connaître la plus petite et la plus grande valeur de la série.

Solution : Pour calculer la plus grande et la plus petite valeur, je pourrais procéder comme pour la moyenne en identifiant les deux variables min et max.

```

1  Variables
2      min: Réel    -- la plus petite des valeurs lues de la série
3      max: Réel    -- la plus grande des valeurs lues de la série

```

Le problème est de savoir avec quelle valeur initialiser max et min. Particulariser une valeur est toujours dangereux. Le mieux est donc de les initialiser avec la première valeur de la série... à condition que cette valeur existe. Le test de la série vide n'est donc plus fait *a posteriori* mais *a priori*.

Les premiers niveaux de raffinement sont alors :

```

1  R0 : Afficher des statistiques sur une série de valeurs réelles
2
3  R1 : Comment « Afficher des statistiques sur une série de valeurs réelles »
4      | Saisir la première valeur x
5      | Si x est nulle Alors
6      |     | Indiquer que les statistiques demandées ne peuvent pas être faites
7      |     Sinon
8      |     | Initialiser les variables statistiques avec x
9      |     | Saisir une nouvelle valeur x
10     |     | TantQue x est NON nulle Faire
11     |     |     | Mettre à jour les variables statistiques
12     |     |     | Saisir une nouvelle valeur x
13     |     | FinTQ
14     |     | Afficher les statistiques
15     | FinSi

```

L'algorithme peut ensuite être le suivant :

```

1  Algorithme statistiques_simples_max
2
3      -- Afficher la moyenne, la plus grande et la plus petite valeur
4      -- d'une série de valeurs réelles lues au clavier.
5      -- La série est terminée par zéro qui n'appartient pas à la série.
6
7  Variables
8      x: Réel    -- un réel lu au clavier
9      nb: Entier -- le nombre de valeurs lues de la série
10     somme: Réel -- la somme des valeurs lues de la série
11     min: Réel  -- la plus petite des valeurs lues de la série
12     max: Réel  -- la plus grande des valeurs lues de la série
13
14  Début
15     -- Saisir le premier réel x
16     Lire(x)
17
18     Si x = 0 Alors
19         -- Indiquer que les statistiques demandées ne peuvent être faites
20         ÉcrireLn("La_série_est_vide.");
21         ÉcrireLn("Les_statistiques_demandées_n'ont_pas_de_sens_!")

```

```

22     Sinon
23         -- Initialiser les variables statistiques avec x
24         max <- x
25         min <- x
26         somme <- x
27         nb <- 1
28
29         -- Saisir une nouvelle valeur x
30         Lire(x)
31
32         TantQue x <> 0 Faire
33             -- Mettre à jour les variables statistiques
34             nb <- nb + 1
35             somme <- somme + x
36             Si x > max Alors
37                 max <- x
38             SinonSi x < min Alors
39                 min <- x
40             FinSi
41
42             -- Saisir une nouvelle valeur x
43             Lire(x)
44         FinTQ
45
46         -- Afficher les statistiques
47         ÉcrireLn("Moyenne_=", somme / nb)
48         ÉcrireLn("Plus_petite_valeur_=", min)
49         ÉcrireLn("Plus_grande_valeur_=", max)
50     FinSi
51 Fin.

1  /*****
2  * Auteur : Xavier Crégut <cregut@enseeiht.fr>
3  * Version : 1.4
4  *
5  * Objectif :
6  *     Afficher la moyenne, la plus grande et la plus petite valeur
7  *     d'une série de valeurs réelles lues au clavier.
8  *
9  *****/
10
11 #include <stdio.h>
12 #include <stdlib.h>
13
14 int main()
15 {
16     double x;           /* un réel lu au clavier */
17     int nb;            /* le nombre de valeurs lues de la série */
18     double somme;      /* la somme des valeurs lues de la série */
19     double min;       /* la plus petite des valeurs lues de la série */
20     double max;       /* la plus grande des valeurs lues de la série */
21

```

```

22  /* afficher la consigne */
23  printf("Donnez une série d'entiers qui se termine par 0.\n");
24
25  /* lire le premier réel */
26  scanf("%lf", &x);
27
28  if (x == 0) {          /* Pas de valeur dans la série */
29      printf("La série est vide.\n");
30      printf("Les statistiques demandées n'ont pas de sens!\n");
31  }
32  else {
33      /* initialiser les variables statistiques avec x */
34      max = x;
35      min = x;
36      somme = x;
37      nb = 1;
38
39      /* lire une nouvelle valeur x */
40      scanf("%lf", &x);
41
42      while (x != 0) {
43          /* mettre à jour les variables statistiques */
44          nb++;
45          somme += x;
46          if (x > max) {
47              max = x;
48          }
49          else if (x < min) {
50              min = x;
51          }
52
53          /* lire une nouvelle valeur x */
54          scanf("%lf", &x);
55      }
56
57      /* afficher les statistiques */
58      printf("Moyenne = %f\n", somme / nb);
59      printf("Plus petite valeur = %f\n", min);
60      printf("Plus grande valeur = %f\n", max);
61  }
62
63  return EXIT_SUCCESS;
64 }
65
66 /*          moyenne, min, max
67  1 2 3 0    -->    2,    1,    3
68  2 -2 0     -->    0,    -2,    2
69  -4 -2 0    -->   -3,    -4,   -2
70  13 0       -->   13,    13,   13
71  0          -->   Non défini
72 */

```

Exercice 3 : Division entière

Étant donnés deux entiers positifs lus au clavier, calculer le quotient et le reste de la division euclidienne du premier nombre par le deuxième. On utilisera uniquement l'addition et la soustraction sur les entiers.

Solution :

Remarque : En utilisant seulement l'addition et la soustraction, on ne peut pas utiliser une boucle **Pour**. Il faut donc choisir entre **TantQue** et **Répéter**.

```

1  Algorithme div_mod
2
3      -- Calculer le quotient (div) et le reste (mod) de la division entière de
4      -- deux entiers lus au clavier
5
6  Variables
7      dividende: Entier    -- dividende lu au clavier, positif
8      diviseur:  Entier    -- diviseur lu au clavier, strictement positif
9      reste:     Entier    -- reste de la division entière
10     quotient:  Entier    -- quotient de la division entière
11
12  Début
13     -- saisir le dividende et le diviseur avec contrôle
14     ...
15
16     -- calculer le quotient et le reste
17     reste <- dividende
18     quotient <- 0
19     TantQue reste >= diviseur Faire
20         { Variant : reste }
21         { Invariant : diviseur * quotient + reste = dividende }
22         quotient <- quotient + 1
23         reste <- reste - diviseur
24     FinTQ
25
26     -- afficher le résultat
27     ÉcrireLn(dividende, "_/_", diviseur, "_=",
28              quotient, "_*__", diviseur, "_+_", reste)
29  Fin.
30
31  --      5      2      -->    2 * 2 + 1
32  --     10      2      -->    5 * 2 + 0
33  --      5      0      -->    diviseur nul !

1  /*****
2  *  Auteur   : Xavier Crégut <cregut@enseeiht.fr>
3  *  Version  : 1.3
4  *  Objectif : Quotient et reste de la division entière
5  *****/
6
7  #include <stdio.h>
8  #include <stdlib.h>
9

```

```
10 /* Calculer le quotient (div) et le reste (mod) de la division entière
11 * de deux entiers lus au clavier.
12 */
13 int main()
14 {
15     int dividende;    /* dividende lu au clavier, positif */
16     int diviseur;    /* diviseur lu au clavier, strictement positif */
17     int reste;       /* reste de la division entière */
18     int quotient;    /* quotient de la division entière */
19
20     /* saisir le dividende et le diviseur */
21     do {
22         printf("Dividende:_");
23         scanf("%d", &dividende);
24         printf("Diviseur:_");
25         scanf("%d", &diviseur);
26
27         /* Vérifier le dividende */
28         if (dividende < 0) {
29             printf("Le_dividende_doit_être_strictement_positif._");
30         }
31
32         /* Vérifier le diviseur */
33         if (diviseur <= 0) {
34             printf("Le_diviseur_doit_être_strictement_positif._");
35         }
36
37     } while (dividende < 0 || diviseur <= 0);
38
39     /* calculer le quotient et le reste */
40     reste = dividende;
41     quotient = 0;
42     while (reste >= diviseur) {
43         /* Variant : reste */
44         /* Invariant : diviseur * quotient + reste = dividende */
45         quotient++;
46         reste = reste - diviseur;
47     }
48
49     /*{ (reste < diviseur) && (diviseur * quotient + reste == dividende) }*/
50
51     /* afficher le résultat */
52     printf("%d/_%d=%d*_%d+_%d\n", dividende, diviseur,
53           quotient, diviseur, reste);
54
55     return EXIT_SUCCESS;
56 }
57
58 /*
59 5      2      --> 2 * 2 + 1
60 2      5      --> 0 * 5 + 2
61 10     2      --> 5 * 2 + 0
```

```
62 0      5      --> 0 * 5 + 0
63 -2     -2     --> dividende non positif, diviseur non > 0
64 -2     5      --> dividende non positif
65 5      -2     --> diviseur non strictement positif
66 5      0      --> Recommencez !
67 */
```

Exercice 4 : Nombres premiers

Écrire un programme qui permet à son utilisateur de saisir une valeur entière et qui, en retour lui indique si c'est un nombre premier ou pas.

Solution : Il s'agit d'afficher si un nombre saisi par l'utilisateur est premier ou non.

1 **R0** : Afficher si un nombre saisi est premier ou non

Pour ce qui concerne les jeux de test, on peut vérifier si le programme fonctionne sur les premiers nombres, par exemple de 1 à 20 ou de 1 à 100.

Le premier niveau de raffinement est classique. Il permet de clairement séparer la partie calcul de la partie interaction avec l'utilisateur.

```

1 R1 : Raffinage De « Afficher si un nombre saisi est premier ou non »
2   | Saisir un nombre                n: out Entier
3   | Déterminer si le nombre est premier  n: in ; premier: out Booléen
4   | Afficher le résultat            n, premier: in

```

Seule la deuxième étape mérite d'être détaillée. Un nombre premier est un nombre qui n'admet pas de diviseurs autres que 1 et lui-même. Nous allons en proposer plusieurs raffinements que nous allons comparer d'un point de vu performance (en nombre d'opérations arithmétiques).

Ainsi, étant donné un entier n , on peut regarder s'il est divisible par les entiers compris entre 2 et $n - 1$. L'idée est d'essayer chaque entier. Si on trouve un diviseur le n n'est pas premier. Si on ne trouve pas de diviseur, n est premier. On se sert donc de la variable booléenne que l'on initialise à *VRAI* et qui sera mise à faux dès que l'on trouve un diviseur. On pourrait donc l'écrire avec un **Pour**.

```

1 premier <- VRAI
2 Pour diviseur <- 1 JusquÀ diviseur = n-1 Faire
3   Si diviseur est un diviseur de n Alors
4     premier <- FAUX
5   FinSi
6 FinPour

```

Ce qui s'écrit de manière équivalente :

```

1 premier <- VRAI
2 Pour diviseur <- 1 JusquÀ diviseur = n-1 Faire
3   premier <- premier Et (diviseur est un diviseur de n)
4 FinPour

```

Si l'utilisation d'un **Pour** est possible, elle est cependant peu judicieuse. En effet, dès qu'on a trouvé un diviseur, on sait que le nombre n n'est pas premier et il est inutile d'essayer d'autres diviseurs. Nous devons donc utiliser un **TantQue** ou un **Répéter**. Nous optons pour un **TantQue**.

Remarquons que nous avons en fait initialisé la variable premier avec $n <> 1$ pour tenir compte du fait que 1 n'est pas un nombre premier.

Voici le raffinement correspondant.

```

1 R2 : Raffinage De « Déterminer si un nombre est premier »
2   | premier <- n > 1
3   | diviseur <- 2
4   | TantQue premier Et (diviseur < n - 1) Faire
5     premier <- Non diviseur est un diviseur de n

```



```

6   |     diviseur <- diviseur + 1
7   | FinTQ
8
9   R3 : Raffinage De « diviseur est un diviseur de n »
10  | Résultat <- n Mod diviseur = 0

1  /*****
2  * Auteur : Xavier Crégut <cregut@enseeiht.fr>
3  * Version : 1.3
4  * Objectif : programme qui demande un entier et indique s'il est premier.
5  *****/
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <stdbool.h>
10
11 int main()
12 {
13     char rep;           /* réponse utilisateur */
14     int nombre;        /* nombre saisi au clavier */
15     bool premier;     /* est-ce que nombre est premier ? */
16     int diviseur;     /* parcourir les diviseurs potentiels de nombre */
17
18     do {
19         /* saisir le nombre */
20         printf("Entrez_un_nombre:_");
21         scanf("%d", &nombre);
22
23         /* déterminer si nombre est premier */
24         premier = nombre != 1;
25         diviseur = 2;
26         while (premier && diviseur < nombre) {
27             premier = nombre % diviseur != 0;
28             diviseur++;
29         }
30
31         /* afficher le résultat */
32         if (premier) {
33             printf("premier");
34         }
35         else {
36             printf("NON_premier");
37         }
38         printf("\n");
39
40         /* demander si l'utilisateur veut recommencez */
41         printf("Recommencer_(o/n)?_");
42         scanf("_%c", &rep);
43     } while (rep != 'n' && rep != 'N');
44
45     return EXIT_SUCCESS;
46 }

```

En fait, on sait qu'il n'est pas nécessaire de regarder les diviseurs au delà de $n/2$ car il ne peut pas y en avoir (trivial). On peut donc reformuler la condition du **TantQue**

```
| TantQue premier Et (diviseur <= n Div 2) Faire
```

Cette optimisation permet de réduire par 2 le nombre d'opérations. Cependant, on peut faire beaucoup mieux. En effet, si un nombre n'est pas premier il admet un diviseur et en fait au moins deux dont l'un au moins est inférieur ou égal à sa racine carrée.

```
| TantQue premier Et (diviseur <= racine carrée de n) Faire
```

Cette optimisation est bien meilleure que la précédente. En effet, si on considère un nombre premier supérieur à 10000, dans la version initiale, on doit considérer 10000 diviseurs, dans la première optimisation 5000 et seulement 100 dans la seconde.

Remarque : Pour éviter d'utiliser les réels et donc les problèmes d'arrondis (et éventuellement une perte de performance), on peut transformer l'expression

```
diviseur <= racine carrée de n
```

par (en élevant au carré)

```
diviseur * diviseur <= n
```

et, pour éviter les débordements (diviseur * diviseur peut dépasser la capacité des entiers), il est préférable de réorganiser les calculs :

```
diviseur <= n Div diviseur
```

Une fois cette transformation réalisée, on obtient donc :

```
| TantQue premier Et (diviseur <= n Div diviseur) Faire
```

On peut encore améliorer l'algorithme. En effet, si on sait que le nombre n'est pas divisible par 2, il est inutile d'essayer les diviseurs pairs (4, 6, 9, etc.). On peut se limiter aux diviseurs impairs (3, 5, 7, 9, 11...). Voici le raffinement correspondant.

```
1 R2 : Raffinage De « Déterminer si un nombre est premier »
2 | Si n = 2 Alors
3 |   premier <- VRAI
4 | Sinon
5 |   | premier <- (n >= 2) Et Non (2 divise n)
6 |   | diviseur <- 3
7 |   | TantQue premier Et (diviseur <= n Div diviseur) Faire
8 |   |   premier <- Non diviseur est un diviseur de n
9 |   |   diviseur <- diviseur + 2
10 |   | FinTQ
11 | FinSi
```

Remarque : On pourrait se passer du **Si** en initialisant premier de la manière suivante :

```
1 premier = (n == 2) Ou ((n >= 3) Et (n Mod 2 <> 0))
```

```
1 /*****
2 * Auteur   : Xavier Crégut <cregut@enseeiht.fr>
3 * Version  : 1.2
4 * Objectif : programme qui demande un entier et indique s'il est premier.
```

```

5  *****/
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <math.h>
10 #include <stdbool.h>
11
12 int main()
13 {
14     char rep;          /* réponse utilisateur */
15     int nombre;       /* nombre saisi au clavier */
16     bool premier;    /* est-ce que nombre est premier ? */
17     int diviseur;    /* parcourir les diviseurs potentiels de nombre */
18
19     do {
20         /* saisir le nombre */
21         printf("Entrez un nombre: ");
22         scanf("%d", &nombre);
23
24         /* déterminer si nombre est premier */
25         premier = (nombre == 2)
26                 || (nombre >= 3 && (nombre % 2 != 0));
27         diviseur = 3;
28         while (premier && diviseur <= nombre / diviseur) {
29             premier = nombre % diviseur != 0;
30             diviseur += 2;
31         }
32
33         /* afficher le résultat */
34         if (premier) {
35             printf("premier");
36         }
37         else {
38             printf("NON_premier");
39         }
40         printf("\n");
41
42         /* demander si l'utilisateur veut recommencez */
43         printf("Recommencer (o/n)? ");
44         scanf("%c", &rep);
45     } while (rep != 'n' && rep != 'N');
46
47     return EXIT_SUCCESS;
48 }

```

Modifier le programme pour qu'il propose à l'utilisateur d'entrer une autre valeur à traiter ou d'arrêter le programme.

Solution : Le raffinement est le suivant :

- 1 **R0** : Indiquer si des entiers saisis au clavier sont premiers ou non
- 2
- 3 **R1** : Raffinage De « R0 »
- 4 | Répéter

```

5      | | Saisir un entier
6      | | Indiquer le caractère premier de l'entier
7      | | Demander à l'utilisateur s'il veut continuer
8      | JusquÀ réponse = 'n'

```

On peut en déduire le raffinement suivant :

```

1  Algorithme nb_premier
2
3      -- Indiquer si un nombre est premier ou non
4      -- Possibilité de recommencer
5
6  Variables
7      n: Entier           -- parcourir les entiers de 1 à max
8      i: Entier           -- parcourir les diviseurs potentiels de n
9      premier: Booléen   -- n est-il premier
10     réponse: Caractère -- réponse de l'utilisateur (o/n)
11
12  Début
13     Répéter             -- analyser un nombre de l'utilisateur
14         -- saisir le nombre
15         ÉcrireLn ("J'indique_si_un_nombre_est_premier")
16         Écrire ("Le_nombre:_")
17         Lire (n)
18
19         -- Déterminer si n est premier
20     Si n <= 3 Alors
21         premier <- n > 0           -- 0 n'est pas premier
22     Sinon
23         premier <- ((n mod 2) <> 0) -- n Non divisible par 2
24             Et ((n mod 3) <> 0)   -- n Non divisible par 3
25         i <- 3
26         TantQue premier Et (i < n Div i) Faire
27             -- n peut encore être premier
28             -- et il reste des diviseurs potentiels
29             i <- i + 2
30             premier <- (n mod i) <> 0   -- n Non divisible par i
31         FinTQ
32     FinSi
33
34     -- afficher le résultat
35     Si premier Alors
36         ÉcrireLn ("OUI, ", n, "_est_premier")
37     Sinon
38         ÉcrireLn ("NON, ", n, "_n'est_pas_premier")
39     FinSi
40
41     -- Demander à l'utilisateur si il veut continuer
42     Écrire ("Encore_un_nombre_(o/n)?_")
43     Lire (réponse)
44     JusquÀ réponse = 'n'
45  Fin.

```

```
1  /*****
2  * Auteur   : Xavier Crégut <cregut@enseeiht.fr>
3  * Version : Revision : 1.2
4  * Objectif : programme qui demande un entier et indique s'il est premier.
5  *****/
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <stdbool.h>
10
11 int main()
12 {
13     char rep; /* réponse utilisateur */
14     int nombre; /* nombre entré au clavier */
15     bool premier; /* est-ce que nombre est premier ? */
16
17     do {
18         /* saisir le nombre */
19         printf("Entrez un nombre : ");
20         scanf("%d", &nombre);
21
22         /* déterminer si nombre est premier */
23         if (nombre == 2) { /* cas de nombre == 2 */
24             premier = true;
25         }
26         else {
27             /* essayer de trouver un diviseur */
28             int diviseur = 1;
29             do {
30                 diviseur++;
31             } while ((nombre % diviseur != 0) && /* pas divisible */
32                    (diviseur < nombre / diviseur)); /* encore possible */
33             premier = (nombre % diviseur) != 0;
34         }
35
36         /* afficher le résultat */
37         if (premier) {
38             printf("premier");
39         }
40         else {
41             printf("NON_premier");
42         }
43         printf("\n");
44
45         /* demander si l'utilisateur veut recommencez */
46         printf("Recommencer (o/n) ? ");
47         scanf("%c", &rep);
48     } while (rep != 'n');
49
50     return EXIT_SUCCESS;
51 }
```

Exercice 5 : Nombres parfaits

Écrire un programme qui affiche tous les nombres parfaits compris entre 2 et 1000.

Un nombre parfait est un entier égal à la somme de ses diviseurs, lui exclu. Par exemple, 28 est un nombre parfait ($28 = 1 + 2 + 4 + 7 + 14$).

Solution :

```

1  R0 : Afficher les nombres parfaits compris entre 2 et Max, lu au clavier
2
3  R1 : Raffinage De « R0 »
4      | Pour n <- 2 JusquÀ n = Max Faire
5      |     | Si n est parfait Alors
6      |     |     | Afficher n
7      |     |     | FinSi
8      |     | FinPour
9
10 R2 : Raffinage De « n est parfait »
11     | Calculer la somme des diviseurs de n (autre que 1 et n)
12     | Résultat <- n = somme des diviseurs
13
14 R3 : Raffinage De « Calculer la somme des diviseurs de n »
15     | somme <- 1
16     | Pour i <- 2 JusquÀ racine carrée de n Faire
17     |     | Si i diviseur de n Alors
18     |     |     | somme <- somme + i + (n Div i)
19     |     |     | FinSi
20     |     | FinPour
21     |     | Si n est un carré parfait Alors
22     |     |     | somme <- somme - racine carrée de n
23     |     | FinSi

1  /*****
2  *  Auteur   : Xavier Crégut <cregut@enseeiht.fr>
3  *  Version  : 1.3
4  *  Objectif : Trouver les nombres parfaits
5  *****/
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <math.h>
10
11
12  int main()
13  {
14      int nombre;
15      for (nombre = 2; nombre <= 1000000; nombre++) {
16          int somme = 1;
17          /* si NB est diviseur alors (nombre / NB) est également
18           * diviseur. Pour trouver l'ensemble des diviseurs, il suffit
19           * donc d'aller jusqu'à sqrt(nombre).
20           */
21          int diviseur = 2;

```

```
22     while (diviseur < nombre / diviseur) {
23         if (nombre % diviseur == 0) {
24             somme = somme + diviseur + nombre / diviseur;
25         }
26         diviseur++;
27     }
28
29     if (diviseur * diviseur == nombre) {    /* nombre est carré parfait */
30         somme += diviseur;
31     }
32
33     if (nombre == somme) {
34         printf("*****_i\n", nombre);
35     }
36 }
37
38 return EXIT_SUCCESS;
39 }
40
41 /* Les premiers nombres parfaits sont : */
42 /* 6 28 496 8128          */
```