N.B. On suppose que tous les tableaux utilisés ont une dimension MAX (constante)

Exercice 1 :

Donner le type et la valeur des expressions suivantes

- *2 + 3 * 4*
- *2.0 + 3 * 4*
- *vrai et (faux ou vrai)*
- *(2 < 3) et (4 > 5)*

Correction :

- *2 + 3 * 4 = 24* → Entier
- *2.0 + 3 * 4 = 24.0* → Réel
- *vrai et (faux ou vrai) = vrai* → Booléen
- *(2 < 3) et (4 > 5) = faux* → Booléen

Exercice 2 :

Donner la table de vérité des expressions booléennes suivantes

- *(a et non b) ou c*
- *(a et non b) ou (non a et b)*

Correction :

a	*b → non b*	*a et non b*	*c*	*(a et non b) ou c*
F	*F → V*	*F*	*V*	*V*
F	*V → F*	*F*	*F*	*F*
V	*F → V*	*V*	*V*	*V*
V	*V → F*	*F*	*F*	*F*

A → non a	*b → non b*	*a et non b*	*non a et b*	*(a et non b) ou (non a et b)*
F → V	*F → V*	*F*	*F*	*F*
F → V	*V → F*	*F*	*V*	*V*
V → F	*F → V*	*V*	*F*	*V*
V → F	*V → F*	*F*	*F*	*F*

Exercice 3 :

Soient *x*, *y*, *z*, *t* quatre variables numériques d'un environnement donné. Exprimé les expressions booléennes correspondant aux situations suivantes :

– Les valeurs de *x* et de *y* sont toutes les deux supérieures à *3*

– Les variables *x*, *y* et *z* sont identiques

– Les valeurs de *x*, *y* et *z* sont identiques mais différentes de celle de *t*

– Les valeurs de *x* est strictement comprise entre les valeurs de *y* et *t*

– Parmi les valeurs de *x*, *y* et *z* deux valeurs au moins sont identiques

– Parmi les valeurs de *x*, *y* et *z* deux valeurs et seulement deux sont identiques

– Parmi les valeurs de x, y et z deux valeurs au plus sont identiques

Correction :

- $x > 3$ et $y > 3$;
- $x = y$ et $y = z$;
- $x = y$ et $y = z$ et $z \neq t$;
- $x > y$ et $x < t$;
- $x = y$ ou $x = z$ ou $y = z$;
- $(x = y$ et $x \neq z)$ ou $(x = z$ et $x \neq y)$ ou $(y = z$ et $x \neq y)$;
- $(x = y$ et $x \neq z)$ ou $(x = z$ et $x \neq y)$ ou $(y = z$ et $x \neq y)$ ou $(x \neq y$ et $y \neq z)$;

Exercice 4 :

Quelles seront les valeurs des variables a et b après exécution des instructions suivantes :

$a \leftarrow 1$;

$b \leftarrow a + 1$;

$a \leftarrow 3$;

Correction :

$a = 3$

$b = 2$

Exercice 5 :

Quelles seront les valeurs des variables *a*, *b* et *c* après exécution des instructions suivantes :

$a \leftarrow 1$;

$b \leftarrow 5$;

$c \leftarrow a - b$;

$a \leftarrow 2$;

$c \leftarrow a + b$;

Correction :

$a = 2$

$b = 5$

$c = 7$

Exercice 6 :

Ecrire un algorithme qui permet d'échanger les valeurs de deux variables entières.

Correction :

c ← a ;

a ← b ;

b ← c ;

Exercice 7 :

Écrire un algorithme qui à partir de trois notes d'un étudiant et de trois coefficients calcule la moyenne.

Correction :

```
Variables       n1, n2, n3, moy : Réel ;
                c1, c2, c3 : Entier ;
début
        écrire(''Donner les trois notes n1, n2, n3 : '') ;
        lire (n1, n2, n3) ;
        écrire(''Donner les trois coefficients c1, c2, c3 : '') ;
        lire (c1, c2, c3) ;
        moy ← (n1 * c1 + n2 * c2 + n3 * c3)/(c1 + c2 + c3) ;
        écrire (''la moyenne = '', moy) ;
fin.
```

Programme C

```
#include <stdio.h>
void main ( ){
        float   n1, n2, n3, moy ;
        int     c1, c2, c3 ;
        printf(''Donner les trois notes n1, n2, n3 :'') ;
        scanf (''%f %f %f'', &n1, &n2, &n3) ;
        printf(''Donner les trois coefficients c1, c2, c3 :'') ;
        scanf (''%d %d %d'', &c1, &c2, &c3) ;
        moy = (n1 * c1 + n2 * c2 + n3 * c3)/(c1 + c2 + c3) ;
        printf (''La moyenne = %f'', moy) ;
}
```

Exercice 8 :

Écrire un algorithme qui à partir d'une somme d'argent donnée, donne le nombre minimal de billets de *50*DH et *20*DH et le nombre de pièces de *2*DH, *1*DH qui la compose.

Correction :

```
Variables       sd, sp, nb50, nb20, np2, np1, d, r : Entier ;
début
        écrire(‘‘La somme à payer par le client : ’’) ;
        lire (sd) ;
        écrire(‘‘La somme donnée par le client : ’’) ;
        lire (sp) ;
        d ← sd – sp ;
        nb50 ← d div 50 ;
        r ← d mod 50 ;
        nb20 ← r div 20 ;
        r ← r mod 20 ;
        np2 ← r div 2 ;
        np1 ← r mod 2 ;
        écrire (‘‘nombre de billets de 50Dh : ’’, nb50, ‘‘ de 20Dh : ’’, nb20) ;
        écrire (‘‘nombre de pièces de 2Dh : ’’, np2,‘‘ de 1Dh : ’’, np1) ;
fin ;
```

Programme C

```
#include <stdio.h>
void main ( ){
        int       sd, sp, nb50, nb20, np2, np1, d, r ;
        écrire(‘‘La somme à payer par le client : ’’) ;
        scanf (‘‘%d’’, &sd) ;
        écrire(‘‘La somme donnée par le client : ’’) ;
        scanf (‘‘%d’’, &sp) ;
        d = sd – sp ;
        nb50 = d/50 ;
        r = d%50 ;
        nb20 = r/20 ;
        r = r%20 ;
        np2 = r/2 ;
        np1 = r%2 ;
        printf (‘‘nombre de billets de 50Dh : %d de 20Dh : %d\n’’, nb50, nb20) ;
        printf (‘‘nombre de pièces de 2Dh : %d de 1Dh : %d\n’’, np2, np1) ;
}
```

Exercice 9 :

Écrire un algorithme qui permet d'effectuer une permutation circulaire des valeurs entières de trois variables *x, y, z* (la valeur de *y* dans *x*, la valeur de *z* dans *y* et la valeur de *x* dans *z*).

Correction :

t ← x ;

x ← y ;

y ← z ;

z ← t ;

Exercice 10 :

Écrire un algorithme qui détermine si une année est bissextile ou non.

On rappelle que les années bissextiles sont multiples de *4*, mais pas de *100*, sauf pour les millénaires qui le sont.

Correction :

```
Variables       a : Entier ;
début
        écrire (''Donner l'année'') ;
        lire (a) ;
        si ( (a mod 4 = 0 et a mod 100 ≠ 0) ou a mod 400 = 0) alors
                écrire (''année bissextile'') ;
        sinon
                écrire (''année non bissextile'') ;
        finsi ;
fin ;
```

Programme C

```
#include <stdio.h>
void main ( ){
        int     a ;
        printf (''Donner l'année'') ;
        scanf (''%d'',&a) ;
        if ( (a % 4 == 0 && a % 100 != 0) || a % 400 == 0)
                printf (''année bissextile'') ;
        else
                printf (''année non bissextile'') ;
}
```

Exercice 11 :

Écrire un algorithme qui détermine le numéro d'un jour dans l'année en fonction du jour, du mois, et de l'année.

Correction :

```
Variables      j, m, a, numj : Entier ;
début
        écrire (''Donner le jour sous forme j/m/a'') ;
        lire (j, m, a) ;
        selon (m)
        debut
                cas 1 : numj ← j ; sortie ;
                cas 2 : numj ← 31+j ; sortie ;
                cas 3 : numj ← 31 + 28+j ; sortie ;
                cas 4 : numj ← 31 + 28+31+j ; sortie ;
                cas 5 : numj ← 31 + 28+31+30+j ; sortie ;
                cas 6 : numj ← 31 + 28+31+30+31+j ; sortie ;
                cas 7 : numj ← 31 + 28+31+30+31+30+j ; sortie ;
                cas 8 : numj ← 31 + 28+31+30+31+30+31+j ; sortie ;
                cas 9 : numj ← 31 + 28+31+30+31+30+31+31+j ; sortie ;
                cas 10 : numj ← 31 + 28+31+30+31+30+31+31+30+j ; sortie ;
                cas 11 : numj ← 31 + 28+31+30+31+30+31+31+30+31+j ; sortie ;
                cas 12 : numj ← 31 + 28+31+30+31+30+31+31+30+31+30+j ;
        finselon ;
        si (m>2 et ( (a mod 4 = 0 et a mod 100 ≠ 0) ou a mod 400 = 0)) alors
                numj ← numj + 1 ;
        finsi ;
        écrire (''Le numéro d'un jour dans l'année'', numj) ;
fin ;
```

Programme C

```
#include <stdio.h>
void main ( ){
        int      j, m, a, numj : Entier ;
        printf (''Donner le jour sous forme j/m/a'') ;
        scanf (''%d/%d/%d'',&j,&m,&a) ;
        switch (m){
                case 1 : numj = j ; break ;
                case 2 : numj = 31+j ; break ;
                case 3 : numj = 31 + 28+j ; break ;
```

```
            case 4 : numj = 31 + 28+31+j ; break ;
            case 5 : numj = 31 + 28+31+30+j ; break ;
            case 6 : numj = 31 + 28+31+30+31+j ; break ;
            case 7 : numj = 31 + 28+31+30+31+30+j ; break ;
            case 8 : numj = 31 + 28+31+30+31+30+31+j ; break ;
            case 9 : numj = 31 + 28+31+30+31+30+31+31+j ; break ;
            case 10 : numj = 31 + 28+31+30+31+30+31+31+30+j ; break ;
            case 11 : numj = 31 + 28+31+30+31+30+31+31+30+31+j ; break ;
            case 12 : numj = 31 + 28+31+30+31+30+31+31+30+31+30+j ;
        }
        if (m>2 && ( (a % 4 == 0 && a % 100 != 0) || a % 400 == 0))
            numj = numj + 1 ;
        printf (‘‘Le numéro d’un jour dans l’année : %d’’,numj) ;
}
```

Exercice 12 :

Afficher la table de conversion entre les degrés Fahrenheit et Celsius de *250* à *−20* degré F par pallier de *10* degrés. On passe de *x* degré F au degré C en calculant (*5/9 x − 160/9*).

Correction :

```
Variables       i : Entier ;
début
        écrire (‘‘Degré Fahrenheit    Degré Celsius’’) ;
        pour i ← -20 à 250 pas de 10 faire
            écrire (i,‘‘       ’’,5/9*i – 160/9) ;
            retourLigne ( ) ;
        finpour ;
fin ;
```

Programme C

```
#include <stdio.h>
void main ( ){
        int      i ;
        printf (‘‘Degré Fahrenheit \t\t Degré Celsius’’) ;
        for (i = -20 ; i <= 250 ; i = i + 10)
            printf (‘‘%d \t\t %f \n’’, i, 5/9*i – 160/9) ;
}
```

Exercice 13 :

Écrire un algorithme qui prend en entrée trois entiers et qui les tri par ordre croissant.

Correction :

```
Variables       a, b, c, temp : Entier ;
début
        lire (a, b, c) ;
        si (a > b) alors
                temp ← a ;
                a ← b ;
                b ← temp ;
        sinon
                si (b > c) alors
                        temp ← b ;
                        b ← c ;
                        c ← temp ;
                finsi ;
        finsi ;
        écrire (a,‘‘ ’’,b,‘‘ ’’,c) ;
fin ;
```

Programme C

```
#include <stdio.h>
void main ( ){
        int     a, b, c, temp ;
        écrire (‘‘Donner la valeur de a, b, c : ’’) ;
        scanf (‘‘%d %d %d’’,&a,&b,&c) ;
        if (a > b){
                temp = a ;
                a = b ;
                b = temp ;
        }
        else
                if (b > c){
                        temp = b ;
                        b = c ;
                        c = temp ;
                }
        printf (‘‘les valeurs dans l’ordre croissant : %d, %d, %d’’, a, b, c) ;
}
```

Exercice 14 :

Écrire un algorithme qui pour un temps donné (représenté sous la forme : heure, minute, seconde) retourne le temps (sous la même représentation) après avoir ajouté une seconde.

Correction :

```
Variables     h, m, s : Entier ;
début
        écrire (‘‘Donner l’heure sous forme h:m:s’’) ;
        lire (h, m, s) ;
        si (s < 59) alors
                s ← s + 1 ;
        sinon
                s ← 0 ;
                si (m < 59) alors
                        m ← m + 1 ;
                sinon
                        m ← 0 ;
                        si (h < 23) alors
                                h ← h + 1 ;
                        sinon
                                h ← 0 ;
                        finsi ;
                finsi ;
        finsi ;
        écrire (h,‘‘ :’’,m,‘‘ :’’,s) ;
fin ;
```

Programme C

```
#include <stdio.h>
void main ( ){
        int     h, m, s ;
        printf (‘‘Donner l’heure sous forme h:m:s’’) ;
        scanf (‘‘%d:%d:%d’’, &h, &m, &s) ;
        if (s < 59)
                s = s + 1 ;
        else{
                s = 0 ;
                if (m < 59)
```

```
                m = m + 1 ;
        else{
                m = 0 ;
                if (h < 23)
                        h = h + 1 ;
                else
                        h = 0 ;
        }
    }
    printf (''l'heure après une seconde : %d:%d:%d'', h, m, s) ;
}
```

Exercice 15 :

Dans une entreprise, le calcul des jours de congés payés s'effectue de la manière suivante : si une personne est entrée dans l'entreprise depuis moins d'un an, elle a droit à deux jours de congés par mois de présence, sinon à *28* jours au moins. Si c'est un cadre et s'il est âgé d'au moins *35* ans et si son ancienneté est supérieure à *3* ans, il lui est accordé *2* jours supplémentaires. S'il est âgé d'au moins *45* ans et si son ancienneté est supérieure à *5* ans, il lui est accordé *4* jours supplémentaires, en plus des *2* accordés pour plus de *35* ans. Écrire un algorithme qui calcule le nombre de jours de congés à partir de l'âge, l'ancienneté et l'appartenance au collège cadre d'un employé.

Correction :

```
Variables    age, anc, nbjc : Entier ;
             cadre : Booléen ;
début
    écrire (''Donner l'âge en année et l'ancienneté en mois :'') ;
    lire (age, anc) ;
    écrire (''Donner le statut de l'employer, si cadre tapez vrai sinon tapez faux :'') ;
    lire (cadre) ;
    si (anc < 12) alors
        nbjc ← anc * 2 ;
    sinon
        nbjc ← 28 ;
    finsi ;
    si (cadre = vrai et age ≥ 35 et anc ≥ 36) alors
        nbjc ← nbjc + 2 ;
    finsi ;
```

```
        si (cadre = vrai et age ≥ 45 et anc ≥ 60) alors
                nbjc ← nbjc + 4 ;
        finsi ;
        écrire (''Nombre de jours de congé ='', nbjc) ;
fin ;
```

Programme C

```
#include <stdio.h>
void main ( ){
        int     age, anc, nbjc, cadre ;
        écrire (''Donner l'âge en année et l'ancienneté en mois :'') ;
        scanf (''%d %d'', &age, &anc) ;
        écrire (''Donner le statut de l'employer, si cadre tapez 1 sinon tapez 0 :'') ;
        scanf (''%d'', &cadre) ;
        if (anc < 12)
                nbjc = anc * 2 ;
        else
                nbjc = 28 ;
        if (cadre == 1 && age >= 35 && anc >= 36)
                nbjc = nbjc + 2 ;
        if (cadre == 1 && age >= 45 && anc >= 60)
                nbjc = nbjc + 4 ;
        printf (''Nombre de jours de congé =%d'', nbjc) ;
}
```

Exercice 16 :

Écrire un algorithme qui effectue la multiplication de deux entiers positifs (notés *x* et *y*) donnés en utilisant uniquement l'addition entière.

Correction :

```
Variables       s, x, y, i : Entier ;
début
        lire (x, y) ;
        s ← 0 ;
        pour i ← 1 à x faire
                s ← s + y ;
        finpour ;
        écrire (s) ;
fin ;
```

Programme C

```
#include <stdio.h>
void main ( ){
        int      s, x, y, i ;
        scanf (‘‘%d %d’’, &x, &y) ;
        s = 0 ;
        for (i = 1 ; i <= x ; i++)
                s = s + y ;
        printf (‘‘%d’’,s) ;
}
```

Exercice 17 :

Créer une variable de type entier *x* et afficher le en binaire en affichant successivement ses bits (du bit de poids faible vers le bit de poids fort). Afficher le aussi en hexa.

Correction :

```
Variables       x, b, x1 : Entier ;
début
        écrire (‘‘Donner la valeur de x et la base b :’’) ;
        lire (x, b) ;
        x1 ← x ;
        tantQue (x1 ≠ 0) faire
                écrire (x1 mod b) ;
                x1 ← x1 div b ;
        fintantQue ;
fin ;
```

Programme C

```
#include <stdio.h>
void main ( ){
        int      x, b, x1 ;
        printf (‘‘Donner la valeur de x et la base b :’’) ;
        scanf (‘‘%d %d’’,&x, &b) ;
        x1 = x ;
        while (x1 != 0){
                printf (‘%d’’, x1%b) ;
                x1 = x1/b ;
        }
}
```

Exercice 18 :

Écrire un programme qui calcule la somme des *1/i* pour *i = 1 à n* pour une valeur de *n* rentrée au clavier. Faire cette somme en partant de *1 à n* et la recalculer en partant de *n* à 1. Comparer ces deux sommes.

Correction :

```
Variables       n, i : Entier ;
                s, p : Réel ;
début
        lire (n) ;
        s ← 0 ;
        p ← 0 ;
        pour i ← 1 à n faire
                s ← s + 1/i ;
                p ← p + 1/(n – i + 1) ;
        finpour ;
        si (p = s) alors
                écrire (‘‘sommes identiques’’) ;
        sinon
                écrire (‘‘sommes différentes’’) ;
        finsi ;
fin ;
```

Programme C

```
#include <stdio.h>
void main ( ){
        int     n, i ;
        double s = 0, p = 0 ;
        scanf (‘‘%d’’, &n) ;
        for (i = 1 ; i <= n ; i++){
                s = s + 1.0/i ;
                p = p + 1.0/(n – i + 1) ;
        }
        if (p == s)
                printf (‘‘sommes identiques’’) ;
        else
                écrire (‘‘sommes différentes’’) ;
}
```

Exercice 19 :

Écrire une procédure qui affiche à l'aide du signe + les figures suivantes :

1- Triangle de hauteur n (ex. *n = 3*) :

```
+
+       +
+       +       +
```

2- Triangle de hauteur n (ex. *n = 3*) :

```
                +
        +       +       +
+       +       +       +       +
```

3- Triangle de hauteur n (ex. *n = 3*) :

```
                +
        +               +
+       +       +       +       +
```

4- Carré de côté n (ex. *n = 3*) :

```
+       +       +
+               +
+       +       +
```

5- Losange de coté n (ex. n=3) :

```
                +
        +               +
+                               +
        +               +
                +
```

N.B : Pour ces exercices, on considère que l'on connait la procédure *retourLigne ( ).*

Correction :

Variables *n, i, j : Entier ;*

début

lire (n) ;

// 1er Triangle

pour i ← 1 à n faire

```
                pour j ← 1 à i faire
                        écrire ('+') ;
                finpour ;
                retourLigne ( ) ;
        finpour ;
// 2ème Triangle
        pour i ← 1 à n faire
                pour j ← 1 à n – i faire
                        écrire (' ') ;
                finpour ;
                pour j ← 1 à 2*i – 1 faire
                        écrire ('+') ;
                finpour ;
                retourLigne ( ) ;
        finpour ;
// 3ème Triangle
        pour i ← 1 à n -1 faire
                écrire (' ') ;
        finpour ;
        écrire ('+') ;
        retourLigne ( ) ;
        pour i ← 2 à n – 1 faire
                pour j ← 1 à n – i faire
                        écrire (' ') ;
                finpour ;
                écrire ('+') ;
                pour j ← 1 à 2*i – 3 faire
                        écrire (' ') ;
                finpour ;
                écrie ('+') ;
                retourLigne ( ) ;
        finpour ;
// Carré
        pour i ← 1 à n faire
                écrire ('+') ;
        finpour ;
        retourLigne ( ) ;
```

```
        pour i ← 2 à n – 1 faire
                écrire ('+') ;
                pour j ← 2 à n – 1 faire
                        écrire (' ') ;
                finpour ;
                écrire ('+') ;
                retourLigne ( ) ;
        finpour ;
        pour i ← 1 à n faire
                écrire ('+') ;
        finpour ;
// Losange
        pour i ← 1 à n – 1 faire
                écrire (' ') ;
        finpour ;
        écrire ('+') ;
        retourLigne ( ) ;
        pour i ← 2 à n faire
                pour j ← 1 à n – i faire
                        écrire (' ') ;
                finpour ;
                écrire ('+') ;
                pour j ← 1 à 2*i – 3 faire
                        écrire (' ') ;
                finpour ;
                écrire ('+') ;
                retourLigne ( ) ;
        finpour ;
        pour i ← 2 à n – 1 faire
                pour j ← 1 à i – 1 faire
                        écrire (' ') ;
                finpour ;
                écrire ('+') ;
                pour j ← 1 à 2* (n – i +1) – 3 faire
                        écrire (' ') ;
                finpour ;
                écrire ('+') ;
```

```
                retourLigne ( ) ;
        finpour ;
        pour i ← 1 à n – 1 faire
                écrire (' ') ;
        finpour ;
        écrire ('+') ;
fin ;
```

Programme C

```
#include <stdio.h>
void main ( ){
        int     n, i, j ;
        scanf (''%d'',&n) ;
// 1er Triangle
        for (i = 1 ; i <= n ; i++) {
                for (j = 1 ; j <= i ; j++)
                        printf (''+'') ;
                printf (''\n'');
        }
// 2ème Triangle
        for (i = 1 ; i <= n ; i++){
                for (j = 1 ; j <= n – 1 ; j++)
                        écrire ('' '') ;
                for (j = 1 ; j <= 2*i – 1 ; j++)
                        printf (''+'') ;
                printf (''\n'') ;
        }
// 3ème Triangle
        for (i = 1 ; i <= n – 1 ; i++)
                printf ('' '') ;
        écrire (''+\n'') ;
        for (i = 2 ; i <= n – 1 ; i++){
                for (j = 1 ; j <= n – i ; j++)
                        printf ('' '') ;
                printf (''+'') ;
                for (j = 1 ; j <= 2*i – 3 ; j++)
                        printf ('' '') ;
                printf (''+\n'') ;
```

```
        }
// Carré
        for (i = 1 ; i <= n ; i++)
            printf (''+'') ;
        printf (''\n'') ;
        for (i = 2 ; i <= n – 1 ; i++){
            printf (''+'') ;
            for (j = 2 ; j <= n – 1 ; j++)
                printf (''  '') ;
            printf (''+\n'') ;
        }
        for (i = 1 ; i <= n ; i++)
            printf (''+'') ;
// Losange
        for (i = 1 ; i <= n – 1 ; i++)
            printf (''  '') ;
        printf (''+\n'') ;
        for (i = 2 ; i <= n ; i++){
            for (j = 1 ; j <= n – i ; j++)
                printf (''  '') ;
            printf (''+'') ;
            for (j = 1 ; j <= 2*i – 3 ; j++)
                printf (''  '') ;
            printf (''+\n'') ;
        }
        for (i = 2 ; i <= n – 1 ; i++){
            for (j = 1 ; j <= i – 1 ; j++)
                printf (''  '') ;
            printf (''+'') ;
            for (j = 1 ; j <= 2* (n – i + 1) – 3 ; j++)
                printf (''  '') ;
            printf (''+\n'') ;
        }
        for (i = 1 ; i <= n – 1 ; i++)
            printf (''  '') ;
        printf (''+'') ;
}
```

Exercice 20 :

Deux nombres entiers n et m sont qualifiés d'amis, si la somme des diviseurs de *n* est égale à *m* et la somme des diviseurs de *m* est égale à *n* (on ne compte pas comme diviseur le nombre lui même et *1*).

Exemple : les nombres *48* et *75* sont deux nombres amis puisque :

Les diviseurs de *48* sont : *2 + 3 + 4 + 6 + 8 + 12 + 16 + 24 = 75*

Les diviseurs de *75* sont : *3 + 5 + 15 + 25 = 48*.

Ecrire un algorithme qui permet de déterminer si deux entiers *n* et *m* sont amis ou non.

Correction :

```
Variables       m, n, sm, sn, j : Entier ;
début
        lire (m, n) ;
        sm ← 0 ;
        sn ← 0 ;
        pour j ← 2 à n div 2 faire
                si (n mod j = 0) alors
                        sn ← sn + j ;
                finsi ;
        finpour ;
        pour j ← 2 à m div 2 faire
                si (m mod j = 0) alors
                        sm ← sm + j ;
                finsi ;
        finpour ;
        si (n = sm et m = sn) alors
                        écrire (n,'' et '',m,'' sont des amis'') ;
        finsi ;
fin ;
```

Programme C

```
#include <stdio.h>
void main ( ){
        int     m, n, sm = 0, sn = 0, j ;
        scanf (''%d %d'', &m, &n) ;
        for (j = 2 ; j <= n % 2 ; j++)
                if (n % j == 0)
                        sn = sn + j ;
```

```
        for (j = 2 ; j <= m % 2 ; j++)
                if (m % j == 0)
                        sm = sm + j ;
        if (n == sm && m == sn)
                        printf (''%d et %d sont des amis'',n, m) ;
        else
                        printf (''%d et %d ne sont pas des amis'',n, m) ;
}
```

Exercice 21 : Développement limité

Lorsque *x* est proche de *0*, *arcsin (x)* peut être approximé à l'aide de la formule suivante :

$$\Sigma\ (1*3*...*(2i-1)*x^{2i+1})/(2^i*(i!)*(2i+1))$$

Écrire un algorithme qui calcule une approximation de *arcsin (x)* de 1 jusqu'au rang *n*.

Correction :

```
Variables       x, r : Réel ;
                n, q, p, s, f, i : Entier ;
début
        lire (x, n) ;
        q ← 1 ; p ← x ; s ← 1 ; f ← 1 ; r ← 0;
        pour i ← 1 à n faire
                q ← q * (2*i – 1) ;
                p ← p * x * x ;
                s ← s * 2 ;
                f ← f * i ;
                r ← r + (q * p)/( p * f * (2*i + 1)) ;
        finpour ;
        écrire (''arcsin('',x,'') = '', r) ;
fin ;
```

Programme C

```
#include <stdio.h>
void main ( ){
        float   x ;
        double r ;
        int     n, q, p, s, f, i ;
        printf (''Donner la valeur de x :'') ;
        scanf (''%f'', &x) ;
```

```
        printf (''Donner l'ordre n :'') ;
        scanf (''%d'', &n) ;
        q = 1 ; p = x ; s = 1 ; f = 1 ; r = 0;
        for (i = 1 ; i <= n ; i++){
                q = q * (2*i – 1) ;
                p = p * x * x ;
                s = s * 2 ;
                f = f * i ;
                r = r + (1.0 * q * p)/( p * f * (2*i + 1)) ;
        }
        printf (''arcsin(%f) = %lf'', x, r) ;
}
```

Exercice 22 :

Calculer le *n* nombre de Fibonacci F_n qui est défini de la manière suivante :

$F_0 = F_1 = 1$

et $F_i = F_{i-1}+F_{i-2}$ pour $i > 1$.

Proposer une variante non récursive utilisant que trois entiers.

Cette dernière technique s'appelle de la programmation dynamique : au lieu de recalculer plusieurs fois la même valeur, on la met dans un tableau. Ici, on n'a même pas besoin de stocker tout le tableau car on n'a besoin que des deux derniers éléments pour calculer la valeur de la case suivante du tableau.

Correction :

```
Variables       F0, F1, F2, i : Entier ;
début
        F0 ← 1 ;
        F1 ← 1 ;
        pour i ← 2 à n faire
                F2 ← F0 + F1 ;
                F0 ← F1 ;
                F1 ← F2 ;
        finpour ;
        écrire (''F'', n, ''='', F2) ;
fin ;
```

Programme C

```
#include <stdio.h>
void main ( ){
```

```
        int     F0, F1, F2, i ;
        F0 ← 1 ;
        F1 ← 1 ;
        for (i = 2 ; i <= n ; i++){
                F2 = F0 + F1 ;
                F0 = F1 ;
                F1 = F2 ;
        }
        printf (''Fibo%d = %d'', n, F2) ;
}
```

<u>*Version récursive*</u>

```
int fibo ( int n){
        i f (n==0 !! n==1)
                return 1 ;
        else
                return fibo (n – 1) + fibo (n – 2) ;
}
void main ( ){
        int     n ;
        scanf(''%d'', &n) ;
        printf (''Fibo%d = %d'', n, fibo(n)) ;
}
```

<u>*Version récursive avec compteur de nombres d'appels*</u>

```
int fibo ( int n){
        static int cpt++ ;
        i f (n==0 !! n==1)
                return 1 ;
        else
                return fibo (n – 1) + fibo (n – 2) ;
}
```

<u>Exercice 23 :</u>

Ecrire une fonction qui à partir d'un réel retourne la valeur absolue de ce réel.

<u>Correction :</u>

```
fonction valeurAbsolue (E a : Réel) : Réel
début
        si (a < 0) alors
```

```
            retourner –a ;
        sinon
            retourner a ;
fin ;
```

Programme C

```
float valeurAbsolue (float a){
        if (a < 0)
            return –a ;
        else
            return a ;
}
```

Exercice 24 :

Ecrire une fonction qui à partir d'un nombre entier strictement positif retourne *VRAI* si ce nombre est pair et *FAUX* sinon.

Correction :

```
fonction parité (E a : Entier) : Booléen
début
        retourner a mod 2 = 0 ;
fin ;
```

Programme C

```
int parité (int a){
        return a % 2 == 0 ;
}
```

Exercice 25 :

Écrire une fonction qui à partir d'un entier strictement positif donné, retourne le résultat booléen *VRAI* ou *FAUX* selon que le nombre est premier ou non. Afficher la liste des nombres premiers inférieurs inférieur à n (ex. *50* : *1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47).*

Correction :

```
fonction estPremier (E a : Entier) : Booléen
Variables       b : Booléen ;
                i : Entier ;
debut
        b ← vrai ;
        i ← 2 ;
        TantQue (b = vrai et i ≤ a div 2) faire
```

```
            si (a mod i = 0) alors
                b ← faux ;
            finsi ;
            i ← i + 1 ;
        fintantQue ;
        retourner b ;
fin ;
procédure nombresPremiers (E n : Entier)
Variables   i : Entier ;
début
        pour i ← 2 à n faire
            si (estPremier(i)) alors
                écrire (i, ‘‘ ’’) ;
            finsi ;
        finpour ;
fin ;
```

Programme C

```
int nombrePremier (int a)
        int     b = 1, i = 2 ;
        while (b = 1 && i <= a/2){
            if (a%i == 0)
                b = 0 ;
            i = i + 1 ;
        }
        return b ;
}
void nombresPremiers (int n){
        int     i ;
        for(i = 2 i <= n ; i++)
            if (estPremier(i))
                printf(‘‘%d ’’, i) ;
}
```

Exercice 26 :

Écrire une procédure qui affiche la suite de tous les nombres parfaits inférieurs ou égaux à un nombre, entier positif donné noté n. Un nombre est dit parfait s’il est égal à la somme de ses diviseurs stricts.

Exemple : *28 = 1 + 2 + 4 + 7 + 14*

Voici la liste des nombres parfaits inférieurs à *10000* : *6, 28, 496, 8128*.

Correction :

```
fonction estParfait (E x : Entier) : Booléen
Variables       s, j : Entier ;
début
        s ← 1 ;
        pour j ← 2 à x div 2 faire
                si (x mod j = 0) alors
                        s ← s + j ;
                finsi ;
        finpour ;
        si (x = s) alors
                retourner  vrai ;
        sinon
                retourner faux ;
        finsi ;
fin ;
procédure nombresParfaits (E n : Entier)
Variables       i : Entier ;
début
        pour i ← 1 à n faire
                si (estParfait(i)) alors
                        écrire (i,‘‘ ’’) ;
                finsi ;
        finpour ;
fin ;
```

Programme C

```
int estParfait (int x){
        int      s, j ;
        s = 1 ;
        for (j = 2 ; j <= x%2 ; j++)
                if (x%j == 0)
                        s = s + j ;
        if (x == s)
                return 1 ;
        else
```

```
        return 0 ;
}
void nombresParfaits (int n){
    int     i ;
    for (i = 1 ; i <= n ; i++){
        if (estParfait(i))
            printf (''%d '',i) ;
    }
}
```

Exercice 27 :

Programmer une procédure qui échange deux éléments.

Correction :

```
procédure échange (E/S a, b : Entier)
    Variables temp : Entier ;
début
    temp ← a ;
    a ← b ;
    b ← temp ;
fin ;
```

Programme C

```
void échange (int* a, int* b){
    int     temp ;
    temp = *a ;
    *a = *b ;
    *b = temp ;
}
```

Exercice 28 :

Lorsque *x* est proche de *0*, *sin (x)* peut être approximé à l'aide de la formule suivante :

$$\Sigma\ (-1)^i * x^{2i+1} / (2i + 1)!$$

Écrire une fonction qui retourne une approximation de *sin (x)* de 1 jusqu'au rang *n*.

Correction 28 :

```
fonction sin (x : Reel, n : Entier) : Réel
variables   i : Entier ;
            num, den, xP, res : Réel ;
```

```
début
        num ← 1 ;
        den ← 1 ;
        xP ← x ;
        res ← 0 ;
        pour i ← 1 à n faire
                num ← – num ;
                den ← den* (2*i + 1)* (2*i) ;
                xP ← x*x*xP ;
                res ← res + num/den *xP ;
        finpour ;
        retourner res ;
fin ;
```

Programme C

```
float calculsin (float x, int n){
        int     i ;
        float   num = 1, den = 1, xP = x, res = 0 ;
        for (i = 1 ; i <= n ; i++){
                num = – num ;
                den = den* (2*i + 1)* (2*i) ;
                xP = x*x*xP ;
                res = res + num/den * xP ;
        }
        return res ;
}
```

Exercice 29 :

Écrire une procédure qui à partir d'un tableau d'entiers *t* d'au moins un entier, fournit le nombre de sous-séquences croissantes de ce tableau, ainsi que les indices de début et de fin de la plus grande sous-séquence.

Par exemple, soit t un tableau de *15* éléments :

1; 2; 5; 3; 12; 25; 13; 8; 4; 7; 24; 28; 32; 11; 14.

Les séquences strictement croissantes sont :

< 1; 2; 5 >;< 3; 12; 25 >;< 13 >;< 8 >;< 4; 7; 24; 28; 32 >;< 11; 14 >.

Le nombre de sous-séquence est : 6 et la plus grande sous-séquence est :

< 4; 7; 24; 28; 32 >.

Correction 30 :

```
procédure sousSequences (E t : tableau[1..MAX] d'Entiers)
Variables       i : Entier ;
début
        cpt ← 0 ;       indD1 ← 0 ;   indF1 ← 0 ;
        i ← 0 ;
        tantQue (i ≤ MAX) faire
                i ← i +1 ;
                écrire (''<'') ;
                indD2 ← i ;
                tantQue (t[i] ≤ t[i + 1] et i ≤ MAX) faire
                        écrire (t[i]) ;
                        i ← i  + 1 ;
                fintantQue ;
                écrire (t[i],''> ;'') ;
                indF2 ← i ;
                si (indF2 – indD2 > indF1 – indD1) alors
                        indD1 ← indD2 ;
                        indF1 ← indF2 ;
                finsi ;
                cpt ← cpt + 1 ;
        fintantQue ;
        écrire (''Le nombre de sous-séquence est : '',cpt) ;
        écrire (''la plus grande sous-séquence est : <'') ;
        pour i ← indD2 à indF2 faire
                écrire (t[i],'','') ;
        finpour ;
        écrire (''>'') ;
fin ;
```

Exercice 31 :

Programmer une fonction qui insère un élément à sa place dans un tableau qu'on supposera déjà trié.

Correction 31 :

```
fonction insertionElement (E/S t : tableau[1..MAX] d'Entier ; E n, x : Entier) : Entier
Variables       i, j : Entier ;
```

```
début
    si (n ≥ MAX) alors
        retourner 0 ;
    sinon
        i ← 1 ;
        tantQue (i ≤ n et x > t[i]) faire
            i ← i + 1 ;
        fintantQue ;
        t[n + 2] ← t[n + 1] ; // transfert d'indicateur de fin du tableau
        si (i ≤ n) alors
            pour j ← i à n faire
                t[n + i – j + 1] ← t[n + i – j] ;
            finpour ;
            t[i] ← x ;
        sinon
            t[n + 1] ← x ;
        finsi ;
    finsi ;
    retourner 1 ;
fin ;
```

Programme C

```
float insererElement (int n, int x, int t[]){
    int     i, j ;
    si (n >= MAX)
        return 0 ;
    else{
        t[n + 1] = t[n] ;       // transfert d'indicateur de fin du tableau d'Entier
        i = 0 ;
        while (i < n && x > t[i])
            i = i + 1 ;
        if (i < n){
            for (j = i ; j < n ; j++)
                t[n + i – j] ← t[n – 1 + i – j] ;
            t[i] ← x ;
        }
        else
```

```
                t[n] ← x ;
            return 1 ;
        }
}
```

Exercice 32 :

Donnez une fonction qui retourne l'indice de la première occurrence de sous chaîne dans une chaîne. Si sous chaîne n'est pas présente dans la chaîne, alors la fonction retourne *-1*.

Correction 32 :

```
fonction sousChaine (E ss : tableau[1..MAX] de caractère ; E s : tableau[1..MAX] de caractère) : Entier
Variables       ls, lss, trouver, i, j : Entier ;
début
        lss ← longueur (ss) ;
        ls ← longueur (s) ;
        trouver ← -1 ;
        i ← 1 ;
        tantQue (trouver = faux et i ≤ ls – lss + 1) faire
                si (ss[1] = s[i]) alors
                        j ← 2 ;
                        tantQue (ss[j] = s[i + j – 1] et j ≤ lss) faire
                                j ← j + 1 ;
                        fintantQue ;
                        si (j > lss) alors
                                trouver ← i ;
                        finsi ;
                finsi ;
                i ← i + 1 ;
        fintantQue ;
        retourner trouver ;
fin ;
```

Programme C

```
int sousChaine (char ss[], char s[]){
        int     lss, ls, i, j, trouver ;
        lss = length (ss) ;
```

```
        ls = length (s) ;
        trouver = -1 ;
        i ← 0 ;
        while (trouver == 0 && i < ls – lss + 1){
                if (ss[0] == s[i]){
                        j = 1 ;
                        while (ss[j] = s[i + j – 1] et j < lss)
                                j = j + 1 ;
                        if (j >= lss)
                                trouver = i ;
                }
                i = i + 1 ;
        }
        return trouver ;
}
```

Exercice 33 :

Soit un tableau *t* d'entiers. Écrire une procédure qui change de place les éléments de ce tableau de telle façon que le nouveau tableau *t* soit une sorte de "miroir" de l'ancien.

Exemple : *1 2 4 6 → 6 4 2 1*

Correction 33 :

```
procédure inversionTab (E/S t : tableau[1..MAX] d'Entier)
Variables       i, temp : Entier ;
début
        pour i ← 1 à MAX div 2 faire
                temp ← t[i] ;
                t[i] ← t[MAX – i +1] ;
                t[MAX – i + 1] ← temp ;
        finpour ;
fin ;
```

Programme C

```
void inversionTab (int t[]){
        int     i, temp ;
        for (i = 0 ; i < MAX/2 ; i++){
                temp = t[i] ;
```

```
            t[i] = t[MAX – i +1] ;
            t[MAX – i + 1] = temp ;
        }
}
```

Exercice 34

Ecrire une fonction qui retourne le minimum et sa position et le maximum et sa position dans un tableau d'entiers.

Correction 34 :

```
procédure minMax (E  t : tableau[1..MAX] d'Entier ; E/S maxi, mini, indMini, indMaxi : Entier)
Variables       i : Entier ;
début
        mini ← t[1] ;
        maxi ← t[1] ;
        indMini ← 1 ;
        indMaxi ← 1 ;
        pour i ← 2 à MAX faire
            si (t[i] < mini) alors
                mini ← t[i] ;
                indMini ← i ;
            sinon
                si (t[i] > maxi) alors
                    maxi ← t[i] ;
                    indMaxi ← i ;
                finsi ;
            finsi ;
        finpour ;
fin ;
```

```
void minMax (int t[], int* maxi, int* mini, int* indMini, int* indMaxi){
        int      i ;
        *mini = t[0] ;
        *maxi = t[0] ;
        *indMini = 0 ;
        *indMaxi = 0 ;
```

```
        for (i = 0 ; i < MAX ; i++){
                if (t[i] < *mini){
                        *min = t[i] ;
                        *indMini = i ;
                }
                else
                        if (t[i] > *maxi){
                                *maxi = t[i] ;
                                *indMaxi = i ;
                        }
        }
}
```

Exercice 35 :

1. Écrire une fonction qui compte le nombre d'occurrences d'un caractère.

2. Écrire une fonction qui permet de savoir si une chaîne de caractères est un palindrome. Par exemple radar, été, rotor.

3- Ecrire une fonction récursive qui permet de savoir si une chaîne est un palindrome.

Correction 35 :

1-

```
fonction occurrence (E c : Caractère ; E ch : tableau[1..MAX] de Caractère) : Entier
Variables       n, i : Entier ;
debut
        n ← 0 ;
        pour i ← 1 à longueur (ch) faire
                si (c = ch[i]) alors
                        n ← n + 1 ;
                finsi ;
        finpour ;
        retourner n ;
fin ;
```

2-

```
fonction palindrome (E ch : tableau[1..MAX] de Caractère) : Boolean
Variables       b : Booléen ;
                l, i : Entier ;
```

```
debut
        b ← vrai ;
        i ← 1 ;
        l ← longueur (ch) ;
        TantQue (b = vrai et i ≤ l div 2) faire
                si (ch[i] ≠ ch[l – i + 1]) alors
                        b ← faux ;
                finsi ;
                i ← i + 1 ;
        fintantQue ;
        retourner b ;
fin ;
```

Programme C

1-

```
int occurrence (char c, char ch[]){
        int     n, i : Entier ;
        n = 0 ;
        for (i = 0 ; i < lentgh (ch) ; i++)
                if (c == ch[i]
                        n++ ;
        return n ;
}
```

2-

```
int palindrome (char ch[]){
        int     l, b = 1, i = 0 ;
        l = lentgh (ch) ;
        while (b == 1 && i < l/2){
                if (ch[i] != ch[l – i – 1])
                        b = 0 ;
                i = i + 1 ;
        }
        return b ;
}
```

Exercice 35 :

Écrire des procédures qui réalisent le tri d'un tableau de *n* entiers par les méthodes du :

1- Le tri par sélection

On cherche d'abord l'élément le plus petit dans le tableau et on l'échange avec le premier élément du tableau; ensuite on cherche l'élément immédiatement supérieur et on l'échange avec le deuxième élément du tableau et ainsi de suite jusqu'à épuisement du tableau.

2- Le tri par insertion

Le tri par insertion consiste à insérer les éléments du tableau un à un de manière à ce qu'à tout instant le tableau soit trié. Pour cela il parcourt les éléments déjà insérées en partant du plus grand et insère l'élément nouveau juste après l'élément immédiatement inférieure qu'il rencontre au cours de son parcours (en début de tri s'il n'en a rencontré aucun).

On peut formaliser cet algorithme sous la forme :

pour chaque i, insérer l'élément t[i] parmi t[1] … t[i – 1]

La procédure d'insertion peut être définie par :

prendre le premier j, en descendant à partir de i, tel que t[j – 1] ≤ t[i], en décalant au fur et à mesure les éléments rencontrés vers la droite, puis insérer t[i] à la j-ième

3- Le tri à bulles

Le tri à bulles consiste à parcourir le tableau et d'échanger deux éléments consécutifs dès qu'ils sont dans l'ordre contraire de l'ordre souhaité. On itère ce parcours jusqu'à ce que le tableau soit trié.

Programme C

```
void echange (int t[], int i, int j){
        int     temp = t[i] ;
        t[i] = t[j] ;
        t[j] = temp ;
}
void triselection (int t[]){
        int     i, j, min ;
        for (i = 0 ; i < MAX – 1 ; i++){
                min = i ;
                for (j = i +1; j < MAX; j++){
                        if (t[j] < t[min])
                                min = j ;
                }
```

```
        echange (t, i, min) ;
        }
}

void triinsertion (int t[]){
        int i, j, v ;
        for ( i = 2 ; i <= MAX; i++){
                v = t[i] ;
                j = i ;
                while (j > 1 && (t[j – 1] > v ) ){
                        t[j] = t[j – 1] ;
                        j = j – 1 ;
                }
                t[j] = v ;
        }
}

void tribulles (int t[]){
        int     i, j;
        for (j = MAX – 1 ; j > 0 ; j++)
                for (i = 0 ; i < MAX – 1; i++)
                        if (t[i] > t[i +1])
                                echange (t, i, i + 1) ;
}
```

Exercice 28 :

En considérant un tableau de trois couleurs ‘A’, ‘B’, ‘C’ (ex. *Bleu, Blanc, Rouge)* réparties aléatoirement de dimension MAX, écrire une procédure qui le tri de telle façon que toutes les couleurs ‘A’ apparaissent au début du tableau, suivies des ‘B’ puis des ‘C’.

Correction 28

```
int estPlusGrand (char x, char y){
        if (x == ‘C’ && y == ‘B’) || (x == ‘C’ et y == ‘A’) || (x == ‘B’ && y == ‘A’)
                return 1 ;
        else
                return 0 ;
}
```

```
void echange (char t[], int i, int j){
        char    temp = t[i] ;
        t[i] = t[j] ;
        t[j] = temp ;
}
void tribulles (char t[]){
        int     i, j;
        for (j = MAX – 1 ; j > 0 ; j++)
                for (i = 0 ; i < MAX – 1; i++)
                        if (estPlusGrand(t[i], t[i +1]))
                                echange (t, i, i + 1) ;
}
```

Exercice :

Ecrire une procédure qui permet de fusionner deux tableaux tries

Programme C

```
void fusion (int a[], int b[], int c[]){
        int     l, i = 0, j = 0, k = 0;
        while(i < MAX1 && j < MAX2){
                if (a[i] <= b [j]){
                        c[k] = a [i] ;
                        i++ ;
                }
                else{
                        c[k] = b[j] ;
                        j++ ;
                }
                k++ ;
        }
        if(i == MAX1)
                for(l = j ; l < MAX2 ; l++){
                        c[k] = b[l] ;
                        k++;
                }
        else
```

```
            for(l = i ; l < MAX1 ; l++){
                c[k] = a[l] ;
                k++;
            }
}
```

Exercice 36 :

Écrire une procédure qui réalise la transposition d'une matrice (en utilisant une seule matrice).

Correction 36 :

```
procédure transposeMatrice (E/S m : matrice[1..MAX, 1..MAX] d'Entier)
Variables       i, j, temp : Entier ;
début
    pour i ← 2 à MAX faire
        pour j ← 1 à i – 1 faire
            temp ← m[i, j] ;
            m[i, j] ← m[j, i] ;
            m[j, i] ← temp ;
        finpour ;
    finpour ;
fin ;
```

Programme C

```
void transposeMatrice (int mat[][MAX]){
    int     i, j, temp ;
    for (i = 1 ; i < MAX , i++)
        for (j = 0 ; j < i – 1 ; j++){
            temp = mat[i][j] ;
            mat[i, j] = mat[j][i] ;
            mat[j][i] = temp ;
        }
}
```

Exercice 37 :

Écrire une procédure qui réalise le produit *C* de deux matrices carrées *A* et *B* de dimension *n*.

Correction 37 :

```
procédure prodMat (E A, B : tableau[1..MAX, 1..MAX] d'Entier, E/S C : tableau[1..MAX, 1..MAX])
Variables       i, j, k : Entier ;
début
        pour i ← 1 à 3 faire
                pour j ← 1 à 3 faire
                        pour k ← 1 à 3 faire
                                C[k, i] ← C[k, i] + A[k, j] * B[j, i] ;
                        finpour ;
                finpour ;
        finpour ;
fin ;
```

Programme C

```
void prodMat (int A[][MAX], int B[][MAX], C[][MAX]){
        int     i, j, k ;
        for (i = 0 ; i < MAX ; i++)
                for (j = 0 ; j < MAX ; j++)
                        for (k = 0 ; k < MAX ; k++)
                                C[k][i] = C[k][i] + A[k][j] * B[j][i] ;
}
```

Exercice 59 :

Programmer l'exponentiation binaire de manière récursive. Cette fonction consiste à calculer x^n en appelant $(x*x)^{n/2}$ si n est pair, et $x*x^{n-1}$ si n est impair et retourne 1 si $n = 0$.

Correction 59 :

```
fonction puissance (E x : Réel ; E n : Entier) : Réel
        si (n = 0) alors
                retourner 1 ;
        retourner x*puissance (x, n – 1) ;
fin ;
```

Programme C

```
float puissance (float x, int n){
        if (n == 0)
                return 1 ;
```

```
        return a*puissance (x, n – 1) ;
}
```

Version 2 :

```
float puissance (float x, int n){
        if (n == 0)
                return 1 ;
        else{
                if (n%2 != 0)
                        return x*puissance (x, n – 1) ;
                else
                        return puissance (x*x, n/2) ;
        }
}
```

Exercice 17 :

Écrire un algorithme qui calcule pour un entier positif donné *n* la valeur de *n!*.

Correction 17 :

```
Variables       n, f, i : Entier ;
début
        lire (n) ;
        f ← 1 ;
        pour i ← 1 à n faire
                f ← f * i ;
        finpour ;
        écrire (f) ;
fin ;
```

Programme C

Méthode itérative

```
#include <stdio.h>
void main ( ){
        int     n, f, i ;
        scanf (''%d'',&n) ;
        f = 1 ;
        for (i = 1 ; i <= n ; i++)
                f = f * i ;
```

```
        printf (''%d'',f) ;
}
```

Méthode récursive

```
int fact ( int n){
        i f (n==0)
                return 1 ;
        else
                return n* fact (n – 1) ;
}
```

Exercice 56 : Algorithme d'Euclide

Programmer le PGCD de deux entiers en utilisant l'algorithme d'Euclide par récursivité.

Méthode

si x > y alors PGCD (x, y) = PGCD (y, x)

si 1 ≤ x ≤ y alors PGCD (x, y) = PGCD (x, y modulo x)

si x = 0 alors PGCD (x, y) = y

Programme C

```
int pgcd (int x, int y){
        i f (x>y )
                return pgcd (y, x) ;
        i f (x>0)
                return pgcd (x, y%x) ;
        else
                return y ;
}
```

Exercice 56 :

Programmer la fonction d'Ackermann définie par :

Ack (0, n) = n + 1

Ack (m, 0) = Ack (m – 1, 1) si n ≥ 1

Ack (m, n) = Ack (m – 1, Ack (m; n – 1)) si n ≥ 1 et m ≥ 1.

Programme C

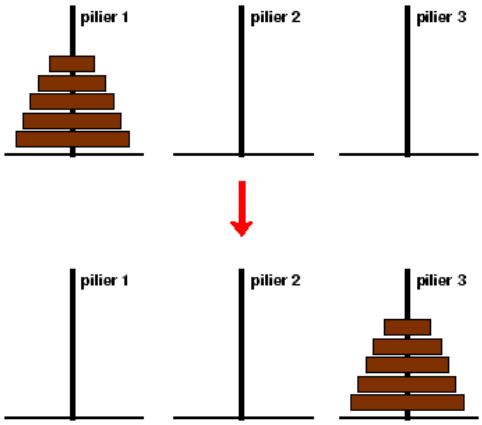
```
int ackerman (int m, int n){
```

```
        if (m == 0)
                return n + 1;
        if (n == 0)
                return ackerman (m – 1 ,1) ;
        return ackerman (m – 1, ackerman (m, n – 1) ) ;
}
```

Exercice 60 : Les tours de Hanoï

Les tours de Hanoï est un jeu solitaire dont l'objectif est de déplacer les disques qui se trouvent sur une tour (par exemple ici la première tour, celle la plus à gauche) vers une autre tour (par exemple la dernière, celle la plus à droite) en suivant les règles suivantes :

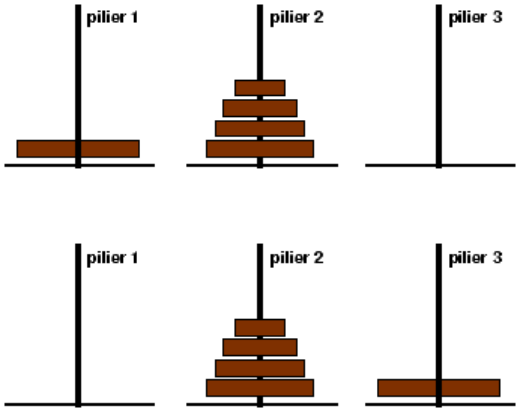
- on ne peut déplacer que le disque se trouvant au sommet d'une tour ;
- on ne peut déplacer qu'un seul disque à la fois ;
- un disque ne peut pas être posé sur un disque plus petit.



Solution au problème

Résoudre le problème des tours de Hanoï, c'est-à-dire déplacer *n* disques d'une tour source '*S*' vers une tour destination '*D*' en utilisant une tour intermédiaire '*I*', revient à :

- déplacer *n – 1* disques de la tour source vers la tour intermédiaire;
- déplacer *1* disque de la tour source vers la tour destination;
- déplacer *n – 1* disques de la tour intermédiaire vers la tour destination.



procédure toursHanoi (E n : Entier ; E s, i, d : Caractère)
début
    si (n = 1) alors
        écrire (s,''→'',d) ;
    sinon
        toursHanoi (n – 1, s, d, i) ;
        toursHanoi (1, s, i, d) ;
        toursHanoi (n – 1, i, s, d) ;
    finsi ;
fin ;

Programme C

```
void toursHanoi (int n, char s, i, d){
    if (n == 1)
        printf (''%c → %c'', s, d) ;
    else{
        toursHanoi (n – 1, s, d, i) ;
        toursHanoi (1, s, i, d) ;
        toursHanoi (n – 1, i, s, d) ;
    }
}
```

Exercice 65 :

Les égyptiens de l'antiquité savaient additionner deux entiers strictement positifs, soustraire 1 à un entier strictement positif, multiplier par *1* et *2* tout entier strictement positif et diviser par *2* un entier strictement positif pair. Voici un exemple qui multiplie *15* par *13* en utilisant uniquement ces opérations :

15 × 13= 15 + 15 × 12

= 15 + 30 × 6

= 15 + 60 × 3

= 75 + 60 × 2

= 75 + 120 × 1

= 75 + 120

= 195

Ecrire l'algorithme récursif qui permet la multiplication de 2 naturels suivant cette méthode.

Correction 66 :

```
fonction multipEgypt (a, b : Entier) : Entier
début
    si (b = 1) alors
        retourner a ;
    sinon
        si (b mod 2 = 0) alors
            retourner multipEgypt (2*a, b div 2) ;
        sinon
            retourner a + multipEgypt (a, b – 1) ;
        finsi ;
    finsi ;
fin ;
```

Programme C

```
int multipEgypt (int a, int b){
    if (b == 1)
        return a ;
    else
        if (b%2 == 0)
            return multipEgypt (2*a, b/2) ;
        else
            return a + multipEgypt (a, b – 1) ;
}
```

Exercice : Recherche dichotomique

Ecrire une fonction qui retourne la position d'un entier dans un tableau trié en utilisant la recherche dichotomique

Programme C

```
int rechercheDico (int x, int i, int j, int t[]){
        int     m ;
        if(i > j)
                return – 1;
        else{
                m = (i + j)/2 ;
                if(t[m] == x)
                        return m;
                else{
                        if (x < t[m])
                                return rechercheDico (x, i, m – 1, t) ;
                        else
                                return rechercheDico (x, m + 1, j, t) ;
                }
        }
}
```

Version itérative

```
int recherche (int x, int t[]){
        int     m, i = 0 , j = MAX – 1 ;
        while (j >=  i){
                m = (i + j)/2 ;
                if (x == t[m])
                        return m ;
                else{
                        if(x < t[m] )
                                j = m – 1 ;
                        else
                                i = m + 1 ;
                }
        }
        return – 1 ;
}
```

Exercice 35 :

Écrire des procédures qui réalisent le tri d'un tableau de *n* entiers par les méthodes du :

1- Le tri par fusion

Etant donné un tableau de taille MAX, on le partage en deux tableaux de taille (approximative) MAX/2 que l'on trie récursivement et que l'on fusionne ensuite. Bien entendu les appels récursifs se terminent lorsque le tableau est de taille 1 auquel cas il est tout trié.
La procédure de tri par fusion d'un tableau est simple.

2- Le tri rapide

Correction 35

Il nous faut utiliser une procédure de fusion de deux sous-tableaux consécutifs d'un même tableau. Le premier sous tableau commence de l'indice mini à m et le deuxième de l'indice m+1 à maxi

```
void fusion (int a[], int b[], int c[]){
    int     l, i = 0, j = 0, k = 0;
    while(i < MAX1 && j < MAX2){
        if (a[i] <= b [j]){
            c[k] = a [i] ;
            i++ ;
        }
        else{
            c[k] = b[j] ;
            j++ ;
        }
        k++ ;
    }
    if(i == MAX1)
        for(l = j ; l < MAX2 ; l++){
            c[k] = b[l] ;
            k++;
        }
    else
        for(l = i ; l < MAX1 ; l++){
            c[k] = a[l] ;
```

```
                k++;
            }
}
```

```
int partition (int t[], int l, int u){
        int     m, pivot, i ;
        m = l ;
        pivot = t[l] ;
        for (i = l +1 ; i <= u ; i++){
            if (t[i] <= pivot){
                echange (t, i, m + 1) ;
                m = m+1;
            }
        echange ( a , l , m) ;
        return m;
}
void trirapide (int t[], int l, int u){
        int m = 0 ;
        if (u > l){
            m = partition (t, l, u) ;
            trirapide (t, l, m –  1) ;
            trirapide (t, m + 1, u) ;
        }
}
```