# Exercices corrigés

17 février 2009

Dans chaque exercice nous proposons un programme toujours structuré de la même manière (cependant certains éléments ne sont pas toujours présents) : #include, #define, définitions de types, prototypes de fonctions ,variables globales, fonction principale, et enfin définitions de fonctions. Cet ordre doit être considéré comme obligatoire. De plus le programme contient des commentaires (/* ... */) et à la fin un exemple d'exécution sous la forme également d'un commentaire (ici chaque ligne commence par //).

**Exercice 1 (Simple et double boucle.)**

1. *Ecrire un programme qui affiche la sortie suivante :*

   *affiche 10 fois 1*
   *affiche 10 fois 2*
   *affiche 10 fois 3*
   *affiche 10 fois 4*
   *affiche 10 fois 5*
   *affiche 10 fois 6*
   *affiche 10 fois 7*
   *affiche 10 fois 8*
   *affiche 10 fois 9*
   *affiche 10 fois 10*
   *affiche 10 fois 11*

2. *Ajouter à ce programme les instructions nécessaires pour que celui-ci affiche les lignes supplémentaires suivantes :*

   *1 1 1 1 1 1 1 1 1 1*
   *2 2 2 2 2 2 2 2 2 2*
   *3 3 3 3 3 3 3 3 3 3*
   *4 4 4 4 4 4 4 4 4 4*
   *5 5 5 5 5 5 5 5 5 5*
   *6 6 6 6 6 6 6 6 6 6*
   *7 7 7 7 7 7 7 7 7 7*
   *8 8 8 8 8 8 8 8 8 8*
   *9 9 9 9 9 9 9 9 9 9*
   *10 10 10 10 10 10 10 10 10 10*
   *11 11 11 11 11 11 11 11 11 11*

**Exercice 2 (Sommes.)**

1. *Ecrire une programme qui affiche la somme des $n$ premiers entiers naturels. La valeur de $n$ est saisie au clavier lors de l'exécution.*
2. *Ecrire un programme qui affiche la somme des entiers compris entre les entiers $d$ et $f$. Les valeurs de $d$ et $f$ sont saisies au clavier lors de l'exécution.*
3. *Ecrire une programme qui affiche la somme des valeurs absolues des entiers compris entre les entiers relatifs $d$ et $f$. Les valeurs de $d$ et $f$ sont saisies au clavier lors de l'exécution.*
4. *Ecrire une programme qui affiche la somme des valeurs absolues des entiers pairs compris entre les entiers relatifs $d$ et $f$. Les valeurs de $d$ et $f$ sont saisies au clavier lors de l'exécution.*

*Note : $n\%2$ est l'expression C dont le résultat est le reste de la division par 2 de $n$.*

**Exercice 3 (Affichage d'un triangle isocèle d'étoiles.)**

*Écrire un programme qui, étant donné un entier naturel impair* `base`*, affiche un triangle isocèle d'étoiles, ayant pour base,* `base` *étoiles. La valeur de la* `base` *sera saisie par l'utilisateur et on considérera qu'il saisit bien un nombre impair. Trois exemples d'exécution sont les suivants :*

```
Nombre d'étoiles à la base du triangle (impair) ?
5
  *
 ***
*****

Nombre d'étoiles à la base du triangle (impair) ?
3
 *
***

Nombre d'étoiles à la base du triangle (impair) ?
1
*
```

**Exercice 4 (Equation du second degré.)**

*Ecrire un programme qui prend en entrée les coefficients d'une équation du second degré et affiche les racines réelles s'il y en a.*

**Exercice 5 (Saisie et affichage d'un tableau.)**

*Ecrire un programme qui saisit un tableau d'entiers de taille $N$ (constante symbolique) et qui l'affiche de telle sorte que tous les entiers pairs se retrouvent*

*avant les entiers impairs. Par exemple, le programme affichera pour un tableau contenant 7 4 7 8 4 6 3 9 6 ses valeurs de la manière suivante : 4 8 4 6 6 7 7 3 9.*

**Exercice 6 (Fonctions qui renvoient ou affichent un résultat.)**
*Écrire un programme qui définit et utilise :*

- *une fonction* fact(n) *qui* renvoie *la factorielle du nombre* n.
- *une fonction* affiche_fact(n) *qui ne* renvoie *rien et* affiche *la factorielle du nombre* n.
- *une fonction* comb (int n , int p) *qui renvoie le nombre de combinaisons de p elements parmi n*
- *une fonction* estDivisible( a, b) *qui renvoie 1 si a est divisible par b*
- *une fonction estPremier(n) qui renvoie 1 si a est premier*

*Rappels :*

- *factorielle(n) = n ! = n (n-1) (n-2) ...1.*
- *un nombre entier n est dit "premier" s'il n'existe aucun entier d dans l'intervalle [2,n ?1] tel que n soit divisible par d.*

**Exercice 7 (Calcul d'une intégrale par la méthode des trapèzes)**

1. *Écrire une fonction* `f` *qui calcule l'inverse* $1/x$ *de son argument.*
2. *Écrire une fonction* `I` *de trois paramètres a, b et n qui calcule une valeur approchée de l'intégrale de la fonction* `f` *entre a et b en utilisant la méthode des trapèzes :*

$$\texttt{I} = \frac{b-a}{n}\left(\frac{\texttt{f}(x_0)}{2} + \texttt{f}(x_1) + \cdots + \texttt{f}(x_{n-1}) + \frac{\texttt{f}(x_n)}{2}\right) \text{ avec } x_i = a + (b-a) \times \frac{i}{n}$$

3. *Écrire un programme qui calcule les valeurs approchées de* `I` *pour* $a = 1$ *et* $b = 2$ *et* $n = 5, 10, 20, 50, 100$ *et compare ces valeurs avec la valeur théorique (*$\ln 2$*).*

*Note : Pour obtenir* $\ln x$ *il faut appeler* `log(x)` *dont le prototype se trouve dans* `math.h`.

**Exercice 8 (Premier nombre premier plus grand que n)** .

1. *Écrire une fonction* `premier` *d'un paramètre entier m et retournant TRUE si le nombre est premier et FALSE dans le cas contraire.*
2. *Écrire une fonction* `prochain_premier` *prenant un paramètre entier* `n` *et retournant le plus petit nombre premier plus grand ou égal à* `n`.
3. *Écrire un programme qui demande un entier n à l'utilisateur et affiche le premier nombre premier plus grand ou égal à n.*

**Exercice 9 (Ecriture d'un entier dans une base.)**

1. *Soit la fonction C suivante :*

```
int decimale( char t[10], int n)
{
        int exposant=0;
        int puissance=1;
        int j;
        int q = n;
        if ( n == 0 )
        {
                t[0]='0';
                return 1;
        }
        else
        {
                while ( puissance <= n )
                {
                        puissance = puissance * 10;
                        exposant = exposant + 1;
                }
                for (j=0; j<exposant; j = j + 1)
                {
                        t[j] = '0' + (q % 10);
                        q=q/10;
                }
                return (exposant);
        }
}
```

   (a) *Quels sont les arguments de la fonction ? Identifiez leurs rôles.*

   (b) *Quelle est la spécificité du passage de tableau comme paramètre d'une fonction.*

   (c) *Quel est le résultat de la fonction ? Quelle est la signification du contenu du tableau* `t` *?*

   (d) *Donnez la signification et le rôle des variables suivantes :* `q` , `puissance` *et* `exposant`.

   (e) *Complétez l'écriture de la fonction avec des commentaires afin de la rendre claire.*

2. (a) *Ecrivez une fonction* `imprime(t,i)` *qui affiche sur une même ligne les* `i` *premiers caractères du tableau* `t`. *Son prototype est :*
   `int imprime ( char t[], int i);`

   (b) *A l'aide des fonctions* `decimale(t,n)` *et* `imprime(t,i)`, *concevez un programme C qui demande à l'utilisateur de saisir un entier positif et qui affiche ses chiffres (dans son écriture décimale) dans l'ordre inverse.*

   *Exemple d'exécution :*

*Entrez un entier positif : 12345*
*54321*

(c) *Modifiez la fonction* `decimale(t,n)` *pour écrire une fonction* `hexadecimale(t,n)`. *Son prototype est :*
`int hexadecimale ( char t[], int i);`
*Ses arguments sont* `t`*, un tableau de char de taille 10, et un entier positif* `n`*. Elle place dans* `t` *les caractères chiffres de son écriture en base 16 et retourne le nombre de chiffres dans son écriture en base 16.*
*Rappel : Les caractères chiffres en base 16 sont :*
*'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A' (10), 'B' (11), 'C' (12), 'D' (13), 'E' (14), 'F' (15).*
*Exemple d'exécution :*
*Entrez un entier positif : 3081*
*90C*

**Exercice 10 (Pi )**

1. *Ecrire une fonction* `cons_rat(n,d)` *qui renvoie le rationnel* $n/d$ *(sous forme d'un* `struct rat`*).*
2. *Ecrire une fonction* `pi_rat(n)` *qui utilise la formule d'Euler* $\pi = 4 + \sum_{i=1}^{\infty} \frac{8}{1-16i^2)}$ *et renvoie l'approximation rationnelle de* $\pi$ *pour une valeur donnée de* $n$.

**Exercice 11 (Mini calculette et complexes)**

*En langage C, les types prédéfinis* float *et* double *permettent de manipuler des nombres à virgule flottante en simple et double précision mais il n'y a pas de type prédéfini permettant de manipuler des nombres complexes. On définira un type* cmplx[1] *en utilisant une structure et deux champs de type double qui représentent la partie réelle et la partie imaginaire.*

*On écrira des fonctions* lit_c(), affiche_c(c), somme_c(c1,c2) *et* produit_c(c1,c2) *que l'on utilisera dans un programme permettant de calculer des expressions de la forme* $((c_1 op_1 c_2) op_2 c_3) op_3 c_4) \dots op_{n-1} c_n)$ *où les complexes sont rentrés dans l'ordre* $c_1, c_2, \dots$ *et l'expression calculée dans l'ordre* $c_1 op_1 c_2$ *d'abord puis son résultat utilisé comme argument gauche de* $op_2$ *et ainsi de suite. L'affichage à l'exécution sera par exemple :*

```
//((a op b) op c)op d ) etc ...
//[Session started at 2006-11-14 15:45:21 +0100.]
//gauche ? (deux réels a et b pour le complexe a+ ib)
//1 1
```

[1] Il serait plus judicieux de nommer ce type cmplx_t.

*//(1.000000,1.000000)*
*//op? (+ ou ∗ ou r ( esultat ) )*
*//+*
*//droit ? (deux réels a et b pour le complexe a+ ib)*
*//3 3*
*//(intermédiaire) ==>(4.000000,4.000000)*
*//op? (+ ou ∗ ou r ( esultat ) )*
*//∗*
*//droit ? (deux réels a et b pour le complexe a+ ib)*
*//2 1*
*//(intermédiaire) ==>(4.000000,12.000000)*
*//op? (+ ou ∗ ou r ( esultat ) )*
*//r*
*//==>(4.000000,12.000000)*

**Exercice 12 (Tri par sélection)**

*Tous les types simples comme les entiers ou les caractères sont munis d'un ordre total permettant de comparer deux éléments. Trier un tableau d'éléments d'un ensemble totalement ordonné consiste à ranger les éléments du tableau de manière croissante par rapport à cet ordre. Il y a plusieurs méthodes de tri, parmi lesquels les tris par sélection et les tris par insertion. Dans ces algorithmes de tri, il est exigé de ne pas utiliser de tableau auxiliaire. Le tableau comportera des entiers positifs avec une valeur négative marquant la fin du tableau.*

*La méthode de tri par sélection "ordinaire" consiste à sélectionner la position du plus petit élément du tableau, à placer cet élément à la première place par échange puis à recommencer sur le reste du tableau. Exemple :*

| | |
|---|---|
| *tableau :* | `\| 12 60 10 25 5` |
| *sélection :* | `\| 12 60 10 25 <5>` |
| *placement :* | `5\| 60 10 25 12` |
| *sélection :* | `5\| 60 <10> 25 12` |
| *placement :* | `5 10\| 60 25 12` |
| *sélection :* | `5 10\| 60 25 <12>` |
| *placement :* | `5 10 12\| 25 60` |
| *sélection :* | `5 10 12\|<25> 60` |
| *placement :* | `5 10 12 25\| 60` |
| *sélection :* | `5 10 12 25\|<60>` |
| *placement :* | `5 10 12 25 60\|` |

1. *Écrire d'abord une fonction qui recherche la position du minimum d'une portion d'un tableau (à partir d'une certain indice* `courant` *jusqu'à un indice* `dernier` *).*
2. *Écrire ensuite une fonction de tri par sélection qui trie le tableau donné en paramètre jusqu'à l'indice* `taille-1`*.*
3. *Écrire un programme qui lit une taille* `taille`*, puis les* `taille` *éléments d'un tableau d'entier à trier, et enfin affiche le tableau trié. La taille maximale du tableau est fixée par une constante* `NMAX` *dans le programme.*

# 1 Correction

### Exercice 1

```
#include <stdlib.h>
#include <stdio.h>
/*      affiche une table de 11 lignes dont la
        ieme ligne contient   iiiiiiiiii   (la valeur de i 10 fois ).
        Dans un premier temps on se contente d'afficher
        " affiche 10 fois i " en passant à la ligne à chaque fois .
        Dans un second temps on remplace cet affichage par une boucle
*/
int main ()
{
        int i;
        int compteur;
        /* version 1 (simple) */
        for(i = 1;i <= 11;i = i + 1)
        {
                printf(" affiche 10 fois %d\n",i);
        }
        /* version 2 (avec une double boucle) */
        for(i = 1;i <= 11;i = i + 1)
        {
                for(compteur = 1;compteur <= 10;compteur = compteur + 1)
                {
                        printf("%d ",i);
                }
                printf("\n");
        }

        return EXIT_SUCCESS;
}
```

### Exercice 2

```
/* déclaration de fonctionalités supplémentaires */
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf */

/* déclaration constantes et types utilisateurs */

/* déclaration de fonctions utilisateurs */

/* fonction principale */
int main()
{
```

```
/* déclaration et initialisation variables */
/* question 1 */
int somme; /* somme à calculer */
int n; /* entier à saisir */
int i; /* var. de boucle */
/* ajout pour questions 2, 3 et 4 */
int d;
int f;

/* 1 : somme des n premiers entiers */
printf("entrez un entier naturel \n");
scanf("%d", &n);
somme = 0; /* élément neutre pour l'addition */

for(i = 1;i <= n;i = i + 1) /* i allant de 1 à n */
{
    /* ajoute i à la somme partielle */
    somme = somme + i;
}
/* i > n */

printf("la somme de %d premiers entiers est %d \n", n, somme);

/* 2 somme des entiers compris entre d et f */
printf("entrez deux entiers relatifs d et f (avec d <= f) \n");
scanf("%d", &d);
scanf("%d", &f);
somme = 0; /* élément neutre pour l'addition */

for(i = d;i <= f;i = i + 1) /* i allant de d à f */
{
    /* ajoute i à la somme partielle */
    somme = somme + i;
}
/* i > f */

printf("la somme des entiers compris entre %d et %d est %d \n", d,f, somme);

/* 3 somme des valeurs absolues des entiers compris entre d et f */
printf("entrez deux entiers relatifs d et f (avec d <= f) \n");
scanf("%d", &d);
scanf("%d", &f);
somme = 0; /* élément neutre pour l'addition */

for(i = d;i <= f;i = i + 1) /* i allant de d à f */
{
    if (i < 0)
    {
        somme = somme - i;
    }
```

```
        else
        {
            somme = somme + i;
        }
    }
    /* i > f */

    printf("la somme des valeurs absolues des entiers compris entre %d et %d est %d \n", d,f, somme);

    /* 4 somme des valeurs absolues des entiers pairs compris entre d et f */
    printf("entrez deux entiers relatifs d et f (avec d <= f) \n");
    scanf("%d", &d);
    scanf("%d", &f);
    somme = 0; /* élément neutre pour l'addition */

    for(i = d;i <= f;i = i + 1) /* i allant de d à f */
    {
        if(i % 2 == 0) /* pair */
        {
            if (i < 0)
            {
                somme = somme - i;
            }
            else
            {
                somme = somme + i;
            }
        }
    }
    /* i > f */

    printf("la somme des valeurs absolues des entiers pairs
    compris entre %d et %d est %d \n", d,f, somme);
    return EXIT_SUCCESS;
}

/* implantation de fonctions utilisateurs */
//td2_L1_06_3 has exited with status 0.
//[Session started at 2006-09-28 11:19:26 +0200.]
//entrez un entier naturel
//3
//la somme de 3 premiers entiers est 6
//entrez deux entiers relatifs d et f (avec d <= f)
//-3 7
//la somme des entiers compris entre -3 et 7 est 22
//entrez deux entiers relatifs d et f (avec d <= f)
//-3 7
//la somme des valeurs absolues des entiers compris entre -3 et 7 est 34
//entrez deux entiers relatifs d et f (avec d <= f)
//la somme des valeurs absolues des entiers pairs compris entre -3 et 7 est 14
```

```
//
//td2_L1_06_3 has exited with status 0.
```

### Exercice 3

```
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf, scanf */

/* déclaration constantes et types utilisateurs */

/* déclaration de fonctions utilisateurs */

int main()
{
    int base; /* un nombre impair d'étoiles à la base du triangle, saisi par l'utilisateur */
    int i; /* var. de boucle */
    int j; /* var. de boucle */

    /* saisie base (impair) */
    printf("Nombre d'etoiles à la base du triangle (impair) ?\n");
    scanf("%d",&base);

    /* affichage triangle isocèle d'étoiles */
    for(i = 1;i <= base;i = i + 2) /* chaque nombre d'étoiles à afficher (base impaire)*/
    {
        /* affiche les blancs */
        for(j = 0;j < (base - i) / 2;j = j + 1) /* (base - i) / 2 fois */
        {
            /* affiche un blanc */
            printf(" ");
        }
        /* j >= (base - i) /2 */

        /* affiche les étoiles */
        for(j = 0;j < i;j = j + 1) /* i fois */
        {
            /* affiche une étoile */
            printf("*");
        }
        /* j >= i */

        /* passe à la ligne suivante */
        printf("\n");
    }
    /* i >= base */

    return EXIT_SUCCESS;
}
```

```
/* implantation de fonctions  utilisateurs  */
```

### Exercice 4

```
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf, scanf */
#include <math.h> /* sqrt */
/* Equations du second degré */
int main () {
        double a,b,c,delta;
        printf("entrez les  coefficients  de l'équation du second degré à résoudre\n");
        scanf("%lf",&a);
        scanf("%lf",&b);
        scanf("%lf",&c);
        delta= (b*b - 4*a*c);
        printf("delta = %f\n", delta);              /* question 1*/
        printf("l'équation %f*xˆ2 + %fx +%f =0",a,b,c);
        if (delta < 0 )
        {
                printf(" n'a pas de racines\n");
        }
        else
        {
                if (delta==0)
                {
                        printf(" a comme racine unique x = %f\n", -b/(2*a));
                }
                else
                {
                        printf(" a comme racines x1 =%f et x2=%f\n",
                                (-b -sqrt(delta))/(2*a),
                                (-b +sqrt(delta))/(2*a)
                        );
                }
        }
        return EXIT_SUCCESS;
}
//[Session started at 2006-10-16 22:56:40 +0200.]
//entrez les  coefficients  de l'équation du second degré à résoudre
//1 0 -1
//delta = 4.000000
//l'équation 1.000000*xˆ2 + 0.000000x +-1.000000 =0 a comme racines x1 =-1.000000 et x2=1.000000
//
//td4_3_L1_06 has exited with status 0.
//[Session started at 2006-10-16 22:56:56 +0200.]
//entrez les  coefficients  de l'équation du second degré à résoudre
//1 2 1
//delta = 0.000000
//l'équation 1.000000*xˆ2 + 2.000000x +1.000000 =0 a comme racine unique x = -1.000000
```

```
//
//td4_3_L1_06 has exited with status 0.
//[Session started at 2006-10-16 22:57:05 +0200.]
//entrez les coefficients de l'équation du second degré à résoudre
//1 2 3
//delta = -8.000000
//l'équation 1.000000*x^2 + 2.000000x +3.000000 =0 n'a pas de racines
//
//td4_3_L1_06 has exited with status 0.
```

### Exercice 5

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

#define N 4
/* lecture d'un tableau et affichage les pairs d'abord, les impairs ensuite */

int main()
{
        int tab[N];
        int i;
        /* lecture */
        for (i = 0; i < N;i = i + 1)
        {
                printf("lecture de l'entier numero %d :", i);
                scanf("%d", &tab[i]);
        }

        /* affichage du tableau dans l'ordre */
        printf("tab = \n{");
        for (i = 0; i< N;i = i + 1)
        {
                printf("%d ", tab[i]);
        }
        printf ("} \n");

        /* affichage du tableau dans l'ordre pairs d'abord */
        /* Attention : une seule instruction dans chacun des for */
        /* donc on peut ne pas utiliser de { } */
        printf("le tableau dans l'ordre <pairs d'abord> = \n{");
        for (i =0; i< N; i = i + 1)
        {
                if(tab[i]%2 == 0)
                {
                        printf ("%d ", tab[i]);
                }
        }
        for (i =0; i< N; i = i + 1)
```

```
        {
                if(tab[i]%2 != 0)
                {
                        printf  ("%d ", tab[i ]);
                }
        }
        printf  ("} \n");

        return EXIT_SUCCESS;
}
//td5.1_ex4 has exited with status 0.
//[Session started at 2006-11-15 11:19:43 +0100.]
//lecture de l'entier numero 0 :1
//lecture de l'entier numero 1 :2
//lecture de l'entier numero 2 :3
//lecture de l'entier numero 3 :4
//tab =
//{1 2 3 4 }
//le tableau dans l'ordre <pairs d'abord> =
//{2 4 1 3 }
```

### Exercice 6

```
#include <stdlib.h>
#include <stdio.h>
/* Ecriture et utilisation de fonctions */

/* prototypes de fonctions */
int somme (int x,int y); /* renvoie la somme de 2 éléments */
int fact (int n); /* n! */
void affiche_fact (int n); /* affichage de n! */
int comb (int n , int p); /* renvoie le nombre de combinaisons de p elements parmi n*/
int estDivisible (int a, int b) ; /* renvoie 1 si a est divisible par b */
int estPremier(int n) ; /* renvoie 1 si a est premier */

/*fonction principale) */
int main ()
{
        int n,p,r;
        /* lectures de n et p */
        printf("entrez un entier\n");
        scanf("%d", &n);
        printf("entrez un entier inférieur au precedent\n");
        scanf("%d", &p);

        /* affichage de la somme de n et p sans la ranger dans une variable */
        printf  ("la somme de %d et %d est %d\n", n,p,somme(n,p));

        /* calcul de la somme de n et p rangement dans r et affichage*/
```

```
        r=somme(n,p);
        printf ("la somme de %d et %d est toujours %d\n", n,p,r);

        /* affichage de fact(n) en utilisant la fonction d'affichage */
        affiche_fact (n);

        /*affichage de fact(n) en appellant fact(n) et en rangeant le résultat */
        r=fact(n);
        printf ("la factorielle de %d est %d \n", n ,r);

        /* affichage du nombre de combinaisons de p objets parmi n */
        printf( "le nombre de façons de prendre %d valeurs (non ordonnées) parmi %d \n", p,n);
        printf ("est %d \n", comb(n,p));

        /* le nombre de combinaisons de p objets parmi n est il premier ? */
        if (estPremier( comb(n,p) ) )
                printf("ce nombre est premier\n");
        else
                printf("ce nombre n'est pas premier\n");
        return EXIT_SUCCESS;


}
 /*définitions des fonctions) */
int somme (int x,int y)
{
        return x + y;
}

int fact (int n)
{
        int i, res = 1;
        for (i = 1; i <=n; i = i + 1)
                res = res *i;
        return res;
}

void affiche_fact (int n)
{
        int i, res = 1;
        for (i = 1; i <=n; i = i + 1)
                res = res *i;
        printf (" %d ! = %d\n",n, res);
        return ;
}

int comb (int n , int p)
{
        return ( fact(n) / ( (fact(p) * fact(n-p)) ) );
}
```

```
int estDivisible(int a, int b)
{
        if (a%b == 0)
                return 1;
        else
                return 0;
}

int estPremier(int n)
{
        int i;
        for ( i =2; i < n-1; i = i + 1)
                if ( estDivisible(n,i))
                        return 0;
        return 1;
}
//entrez un entier inférieur au precedent
//4
//la somme de 5 et 4 est 9
//la somme de 5 et 4 est toujours 9
// 5 ! = 120
//la factorielle de 5 est 120
//le nombre de façons de prendre 4 valeurs (non ordonnées) parmi 5
//est 5
//ce nombre est premier
//
//td5.2_ex1_2_3 has exited with status 0.
```

**Exercice 7**

```
/* Programme 3 : intégration avec la méthode des trapezes */
#include <stdlib.h>
#include <stdio.h>
/* Comme on utilise la fonction log, il faut inclure math.h */
#include <math.h>

/* Fonction à intégrer */
double f(double x)
{
    return 1/x;
}

/* Calcul de l'integrale de f de a à b */
double I(double a, double b, int n)
{
    double s;
    int i;
    s = (f(a) + f(b))/2;
    for(i = 1; i < n; i = i + 1)
        s = s + f(a + (b-a)*i/n);
```

```
    return s*(b-a)/n;
}

/* Calcul de I pour un n particulier */
void approcheLn2(int n)
{
    double x, y, erreur;
    x = I(1, 2, n);    /* La valeur approchée pour n */
    y = log(2);        /* La valeur réelle */
    erreur = (x-y) / y * 100;
    erreur = (erreur >= 0) ? erreur : - erreur;
    printf(" Pour N = %d : I = %g L'erreur avec ln2 est de %g%%\n", n, x, erreur);
}

/* Fonction principale */
void main(void)
{
     approcheLn2(5);
     approcheLn2(10);
     approcheLn2(20);
     approcheLn2(50);
     approcheLn2(100);
     return EXIT_SUCCESS;
}
```

**Exercice 8**

```
/* Programme : le prochain premier */
#include <stdlib.h>
#include <stdio.h>

#define TRUE 1
#define FALSE 0

/* Fonction testant si un nombre est premier */
int premier(int m)
{
    int i;
    for(i = 2; i < m; i = i + 1)
        if ((m % i) == 0) /* divisible */
            return FALSE;
    return TRUE;
}

/* Fonction cherchant le premier nb premier plus grand que n */
int prochain_premier(int n)
{
    while(! premier(n))
        n = n + 1;
    return n;
}
```

```
/* La fonction principale */
void main(void)
{
    int k;
    printf("Entrez un entier  positif  ");
    scanf("%d", &k);
    printf("Le prcohain nombre premier de %d est %d\n", k, prochain_premier(k));
    return EXIT_SUCCESS;
}
```

### Exercice 9

La spécificité du tableau lorsqu'il est passé en argument est que le tableau peut-être modifié dans la fonction et que la modification persiste au retour de la fonction (ce qui est passé n'est pas une copie du tableau mais une indication de sa position dans la mémoire : son *adresse*). Ci-dessous cela est utilisé dans la fonction `décimale` qui renvoie une résultat mais modifie également le tableau `t` passé en argument.

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

/* decimale( t,n) remplit le tableau de caractères t en t[0], t[1], ... t[e-1]
et renvoie e, tels que n= t_0*10^0 + a_1*10^1 + ... + a_(e-1) *10^(e-1) */
int decimale( char t[10], int n);
/* affiche sur une même ligne les k premiers caractères du tableau t */
int imprime (char t[10], int k);
/* hexadecimale( t,n) remplit le tableau de caractères t en t[0], t[1], ... t[e-1]
et renvoie e, tels que n= t_0*16^0 + a_1*16^1 + ... + a_(e-1) *16^(e-1) */
int hexadecimale( char t[10], int n);

int main () {
    // insert code here ...
        char ch[10];
        int n,e;
        /* saisit un entier n et affiche sa representation décimale t[0], t[1],  t[e-1]*/
        /* rangée dans le tableau t.
        On a ainsi n= t_0*10^0 + a_1*10^1 + ... + a_(e-1) *10^(e-1) */
    printf(" nombre \n");

        scanf("%d",&n);
        e=decimale(ch,n);
        printf("premier exposant e tel que 10^e > %d est %d\n", n, e);
        printf("représentation décimale de %d en a_0*10^10 + a_1*10^1 + ... \n");
        imprime(ch,e);
        e=hexadecimale(ch,n);
        printf("premier exposant e tel que 16^e > %d est %d\n", n, e);
        printf("représentation hexadécimale de %d en a_0*16^10 + a_1*16^1 + ... \n");
```

```
        imprime(ch,e);
    return EXIT_SUCCESS;
}

/* decimale( t,n) remplit le tableau de caractères t en t[0], t[1], ... t[e-1]
et renvoie e, tels que n= t_0*10^0 + a_1*10^1 + ... + a_(e-1) *10^(e-1) */
int decimale( char t[10], int n)
{
        /* On cherche le plus petit exposant e tel que puissance = 10^e > n */
        /* Dans ce cas on peut écrire
              n= t_0*10^0 + a_1*10^1 + ... + a_(e-1) *10^(e-1) )*/
        /* on écrit les coeeficients a_0, ... , a_(e-1)
             dans le tableau de caractères t[0], ..., t[e-1] */
        int exposant=0;
        int puissance=1;
        int j;
        int q = n;
        if ( n == 0 )
        {
                t[0]='0'; /* 0= 0*10^0 */
                return 1; /* 10^0 = 1 > 0 */
        }
        else
        {
                /* on cherche puissance et exposant tels que
                puissance=10^exposant > n */
                while ( puissance <= n )
                {
                        puissance = puissance * 10;
                        exposant = exposant + 1;
                }
                /* On écrit les a_j de la représentation dans t
                   (entre 0 et exposant-1) : */
                /* par exemple : 153 % 10 = 3 et 153 / 10 = 15 puis
                15 % 10 = 5 et 15 /10 = 1
                 puis 1 % 10 = 1 et 1 / 10 = 0 ===> t= 3 5 1*/
                for (j=0; j<exposant; j = j + 1)
                {
                        t[j] = '0' + (q % 10);
                        q=q/10;
                }
                return (exposant);
        }
}
int imprime (char t[10], int k){
        int j;
        for (j=0; j<k; j = j + 1)
        {
                printf("%c ", t[j]);
        }
```

```
        printf("\n");
    return 0; /* valeur de retour à ignorer */
}
/* hexadecimale( t,n) remplit le tableau de caractères t en t[0], t[1], ... t[e-1]
et renvoie e, tels que n= t_0*16^0 + a_1*16^1 + ... + a_(e-1) *16^(e-1) */
int hexadecimale( char t[10], int n)
{
        /* On cherche le plus petit exposant e tel que puissance = 16^e > n */
        /* Dans ce cas on peut écrire
            n= t_0*16^0 + a_1*16^1 + ... + a_(e-1) *16^(e-1) )*/
        /* on écrit les coeeficients a_0, ... , a_(e-1)
            dans le tableau de caractères t[0], ..., t[e-1] */
        int exposant=0;
        int puissance=1;
        int j;
        char tab[16] ="0123456789ABCDEF" ;
        int q = n;
        if ( n == 0 )
        {
                t[0]='0'; /* 0= 0*16^0 */
                return 1; /* 16^0 = 1 > 0 */
        }
        else
        {
                /* on cherche puissance et exposant tels que
                    puissance=16^exposant > n */
                while ( puissance <= n )
                {
                        puissance = puissance * 16;
                        exposant = exposant + 1;
                }
                /* On écrit les a_j de la représentation dans t
                   (entre 0 et exposant-1) : */
                for (j=0; j<exposant; j = j + 1)
                {
                        t[j] = tab[q % 16];
                        q=q/16;
                }
                return (exposant);
        }
}

//nombre
//3081
//premier exposant e tel que 10^e > 3081 est 4
//représentation décimale de 0 en a_0*10^10 + a_1*10^1 + ...
//1 8 0 3
//premier exposant e tel que 16^e > 3081 est 3
//représentation hexadécimale de 0 en a_0*16^10 + a_1*16^1 + ...
//9 0 C
```

**Exercice 10**

Il faut un programme avec toutes les fonctions précédentes. Il appellera `pi_rat(n)` qui renvoie le rationnel et l'affichera.

```
  struct rat
  {
    int num;
    int den;
  }

/* on utilise une fonction struct rat cons_rat(int n,int d) qui renvoie le rationnel n/d*/
 struct rat cons_rat(int n,int d)
{
   struct rat r;
   r.num =n;
   r.den =d;
   return r;
 }

 /* on utilise une fonction struct rat somme_rat(struct rat c1, struct rat c2) qui renvoie la somme des deux ratio
 struct rat somme_rat(struct rat c1, struct rat c2)
{
        struct rat c;
        c.num=c1.num*c2.den + c2.num*c1.den ;
        c.den=c1.den*c2.den;
        return c;
}

struct rat pri_rat(int n){
    int i;
    struct rat res, ri;
    res =cons_rat(4,1);
    for (i=1;i<=n; i = i + 1)
    {
        ri  = cons_rat(8,1-16i*i)
        res  = somme_rat(res, ri);
    }
    return res;
 }
```

**Exercice 11**

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

#define N 5

/* type */
struct cmplx
```

```
{
        double re;
        double im;
};

/* prototypes */
struct cmplx lit_c(void);
void affiche_c (struct cmplx c);
struct cmplx somme_c(struct cmplx c1, struct cmplx c2);
struct cmplx produit_c(struct cmplx c1, struct cmplx c2);

/* principale */
int main()
{
        struct cmplx resg,d;
        char rep[10];
        printf("gauche ? (deux réels a et b pour le complexe a+ ib)\n");
        resg=lit_c ();
        affiche_c (resg);
        printf("op? (+ ou * ou r (esultat) ) \n");
        scanf("%s",rep);
        while (rep[0] !='r')
        {
                printf("droit ? (deux réels a et b pour le complexe a+ ib)\n");
                d=lit_c ();
                if (rep[0]=='+')
                {
                        resg=somme_c(resg,d);
                }
                else
                {
                        if (rep[0]=='*')
                        {
                                resg=produit_c(resg,d);
                        }
                        else
                        {
                                printf("erreur\n");break;
                        }
                }
                printf("(intermédiaire) ==>");
                affiche_c (resg);
                printf("op? (+ ou * ou r (esultat) ) \n");
                scanf("%s",rep);
        }
        printf("==>");
        affiche_c (resg);
        return EXIT_SUCCESS;
}
/*fonctions */
```

```
struct cmplx lit_c(void)
{
        struct cmplx c;
        scanf("%lf",&c.re);
        scanf("%lf",&c.im);
        return (c);
}
void affiche_c (struct cmplx c)
{
        printf("(%lf,%lf)\n",c.re,c.im);
        return ;
}
struct cmplx somme_c(struct cmplx c1, struct cmplx c2)
{
        struct cmplx c;
        c.re=c1.re+c2.re;
        c.im=c1.im+c2.im;
        return (c);
}
struct cmplx produit_c(struct cmplx c1, struct cmplx c2)
{
        struct cmplx c;
        c.re=(c1.re*c2.re)-(c1.im*c2.im);
        c.im=(c1.re*c2.im)+(c1.im*c2.re);
        return (c);
}
//((a op b) op c)op d ) etc ...
//[Session started at 2006-11-14 15:45:21 +0100.]
//gauche ? (deux réels a et b pour le complexe a+ ib)
//1 1
//(1.000000,1.000000)
//op? (+ ou * ou r (esultat) )
//+
//droit ? (deux réels a et b pour le complexe a+ ib)
//3 3
//(intermédiaire) ==>(4.000000,4.000000)
//op? (+ ou * ou r (esultat) )
//*
//droit ? (deux réels a et b pour le complexe a+ ib)
//2 1
//(intermédiaire) ==>(4.000000,12.000000)
//op? (+ ou * ou r (esultat) )
//r
//==>(4.000000,12.000000)
```

**Exercice 12** `#include <stdlib.h>`
`#include <stdio.h>`

`#define NMAX 1000`

```
void triParSelection( int tab [], int taille );
int posMin(int tab[], int courant, int taille );

int main ()
{
        int i, taille ;
        int t[NMAX];
        // insert code here ...
        printf(" taille ? ");
        scanf("%d",&taille);
        for (i=0; i< taille ; i=i+1)
        {
                scanf("%d",&t[i]);
        }

        triParSelection (t, taille );
        for (i=0; i< taille ; i=i+1)
        {
                printf("%d ",t[i ]);
        }
        printf("\n");

        return EXIT_SUCCESS;
}
int posMin(int tab[], int courant, int taille )
{
        int i;
        int pmin=courant;
        for (i=courant+1; i< taille; i=i+1){
                if (tab[i] < tab[pmin])
                        pmin=i;
                /* pmin est la position du plus petit élément entre courant et i */
        }
        /* pmin est la position du plus petit élément entre courant et taille -1 */
        return pmin;
}
void triParSelection( int tab [], int taille )
{
        int i,p;
        int aux;
        for (i=0; i< taille ; i=i+1){
                p=posMin(tab,i, taille );
                aux=tab[i];
                tab[i]=tab[p];
                tab[p]=aux;
                /* tab est ordonné jusqu'à la position i incluse et contient entre 0 et i
                les plus petits éléments du tableau */
        }
        /* tab est ordonné jusqu'à la position taille -1 */
```

```
        /* on peut éviter une itération, comment ?*/
}
```

//[Session started at 2009-02-14 13 :00 :14 +0100.] //taille ? 4 //1 //3 //2 //0
//0 1 2 3