# Exercices de langage C

L1 MASS, année 2007–2008

# TD 1 : Affichage et variables

Exercice 1. Écrire un programme de langage C qui affiche le message suivant :

```
Bonjour, entrer un nombre entier s'il vous plait :
```

L'utilisateur doit alors entrer un nombre entier, (par exemple 72), et le programme doit répondre :

```
Le nombre que vous avez entre' est 72
```

Corrigé : Pour le premier exercice, un corrigé un peu long qui fait le point sur diverses questions.

```
#include<stdio.h>

main(){
  int n;

  printf(
    "Bonjour, entrer un nombre entier s'il vous plait :");
  scanf("%i", &n);
  printf("\n");
  printf("Le nombre que vous avez entre' est %i.\n", n);
}
```

- **Travail en salle machines :**
  Il a été dit en cours qu'on travaille sous linux. Rappel : pour éditer, choisir Emacs ou tout autre éditeur dans le menu "chapeau". J'aime bien Emacs, car en C la coloration syntaxique et l'indentation automatiques sont très pratiques. En plus, il n'y a pas besoin de connaître des raccourcis clavier abscons comme le croit parfois : l'essentiel est dans les menus déroulant. Avec CTRL-X 2 on coupe la fenêtre en deux, avec ESC-X SHELL, on lance un shell dans emacs, ce qui fait qu'on peut tout faire dans Emacs. Personnellement, je ne montrerai pas ces fonctionnalités aux étudiants au début.
- **Login, compilation, ...**
  Login pour tous les étudiants : user, pas de mot de passe.
  Pour compiler et exécuter :
  ```
  gcc prog.c -o prog
  ./prog
  ```

Attention, en cours je n'ai pas parlé de ls, cd, mkdir etc. Si le groupe avance vite ça peut meubler. J'en parlerai en cours à la 2$^{e}$ou 3$^{e}$séance. Bien vérifier que les étudiants font la différence entre éditeur de texte (à opposer à traitement de texte) et fenêtre de commande (= shell).

- **Corrigé de l'exercice proprement dit**
  Cet exercice a été traité en cours, il n'y a donc pas à le corriger à proprement parler. C'est l'occasion de prendre en main l'interface, d'apprendre à compiler, etc. Une remarque très importante cependant : il faut demander aux étudiants, surtout aux plus faibles de ne pas tout taper d'un coup. Commencer d'abord par afficher bonjour, puis seulement, taper le scanf, ...
  Remarque : j'utilise %i pour afficher les int, et je réserve %u pour les unsigned int. Je pense que c'est prévu comme ça en C, mais je n'ai pas de religion bien stricte sur la question.
- **Comportement des étudiants**
  Cet exercice est aussi l'occasion de vérifier que les étudiants ont bien le cours avec eux, les exercices préparés, et de les engueuler dans le cas contraire ! M'avertir rapidement s'ils ne travaillent pas ou en cas de problèmes de comportement en TD.

Exercice 2. 1. Écrire un programme qui demande à l'utilisateur deux nombres entiers et qui affiche les résultats des opérations suivantes appliquées à ces deux entiers : addition, soustraction, multiplication, division euclidienne.

2. Que se passe-t-il pour l'addition si on demande les nombres 2 147 483 647 et 1 ? Expliquer.

3. Comment faire pour afficher le résultat de la division des deux nombres entiers (dont le résultat n'est pas forcément entier) ?

Corrigé :

```
#include<stdio.h>

main()
{
  int a, b;

  printf("Bonjour, entrez deux nombres a et b svp: ");
  scanf("%i%i", &a, &b);
  printf("a+b vaut %i\n", a+b);
  printf("a-b vaut %i\n", a-b);
  printf("axb vaut %i\n", a*b);
  printf("Le quotient dans la division "
```

```
        "euclidienne de a par b "
        "vaut %i\n", a/b);
  printf("Le reste dans la division "
        "euclidienne de a par b "
        "vaut %i\n", a%b);


  // Pour avoir la division reelle et non plus euclidienne :

  printf("Le quotient de a par b "
        "vaut %f\n", a/(float)b);

  // Solution plus simple a montrer aux etudiants
  // tout passer en float :

  float c, d;

  printf("Bonjour, entrez deux nombres a et b svp: ");
  scanf("%f%f", &c, &d);
  printf("a+b vaut %f\n", c+d);
  printf("a-b vaut %f\n", c-d);
  printf("axb vaut %f\n", c*d);
  printf("a/b vaut %f",   a/b);
}
```

Attention, mathématiquement la division euclidienne a deux "résultats" : le quotient et le reste. Le cast n'a pas été vu en cours pour le moment. Il faut ou bien l'expliquer, ou bien tout passer en float. Pour la deuxième question, on dépasse la capacité du type int. Attention, pour produire l'effet désiré, il faut bien utiliser %i et non pas %u.

Exercice 3. On considère le programme en langage C suivant :

```
#include<stdio.h>

main()
{
  int p, s, n;

  // p comme precedent
  // s comme suivant

  //Point d'observation 1
```

```
  printf("%i\n", n);

  p=1; s=1;

  //Point d'observation 2

  n=s; s=p+s; p=n; //Point d'obervation 3
  n=s; s=p+s; p=n; //Point d'obervation 4
  n=s; s=p+s; p=n; //Point d'obervation 5
  n=s; s=p+s; p=n; //Point d'obervation 6
  n=s; s=p+s; p=n; //Point d'obervation 7


  printf("Le septieme terme de la suite de Fibonacci vaut %i\n", s);
}
```

1. Tracer l'exécution du programme.

2. Vérifier en tapant et en exécutant le programme.

Corrigé :

D'après le cours, tracer l'exécution d'un programme, c'est : à chaque point d'observation donner les valeurs de toutes les variables. Quand on trace un programme, on ne se soucie pas de ce qui est affiché par les printf. Afin d'uniformiser tout ça et de se préparer à des exercices plus difficiles, il souhaitable que les étudiants adoptent la présentation suivante :

| Point d'observation | p | s | n |
|---|---|---|---|
| 1 | Indéterminé | Indéterminé | Indéterminé |
| 2 | 1 | 1 | Indéterminé |
| 3 | 1 | 2 | 1 |
| 4 | 2 | 3 | 2 |
| 5 | 3 | 5 | 3 |
| 6 | 5 | 8 | 5 |
| 7 | 8 | 13 | 8 |

A priori les étudiants ne connaissent pas la suite de Fibonacci : $f_0 = f_1 = 1$, pour tout $n \geq 0$, $f_{n+2} = f_n + f_{n+1}$. Les boucles n'ont pas été vues au premier cours. Cet exercice est l'occasion de dire que quelque chose ne va pas dans ce programme (comment calculer $f_{500}$ ?), et qu'au prochain cours, on verra comment faire ça mieux.

Exercice 4. On s'intéresse au problème suivant : comment faire pour échanger la valeur de deux ou plusieurs variables ?

1. Tracer l'exécution du programme suivant :

```
#include<stdio.h>

main()
{
  int a, b;

  a=3; b=5; //Point d'observation 1

  printf("a vaut %i\n", a);
  printf("b vaut %i\n", b);

  a=b; b=a; //Point d'observation 2

  printf("a vaut %i\n", a);
  printf("b vaut %i\n", b);
}
```

Les valeurs de $a$ et $b$ ont-elles été échangées ?

2. Proposer un programme pour échanger les valeurs de $a$ et $b$ à l'aide d'une troisième variable $t$.

3. Peut-on se passer d'une troisième variable ?

4. Écrire un programme qui permute les valeurs de $a$, $b$ et $c$. C'est-à-dire qu'après l'exécution du programme, $a$ doit avoir la valeur de $b$, $b$ celle de $c$ et $c$ celle de $a$.

Corrigé :

1. Évidemment, le programme ne marche pas ...

| Point d'observation | a | b |
|---|---|---|
| 1 | 3 | 5 |
| 2 | 5 | 5 |

2. Cette question est l'occasion d'aborder la question des commentaires. DONNEZ AUX ETUDIANTS L'HABITUDE DE COMMENTER LEURS PROGRAMMES.

```
#include<stdio.h>

main()
{
  int a, b;
  int t; //Variable temporaire
```

```
  a=3; b=5;

  printf("a vaut %i\n", a);
  printf("b vaut %i\n", b);

  t=a; //On sauvegarde la valeur de a dans la variable temporaire
  a=b;
  b=t; //On recupere la valeur de a

  printf("a vaut %i\n", a);
  printf("b vaut %i\n", b);
}
```

3. Oui, on peut s'en passer :

```
  a = a + b;
  b = a - b;
  a = a - b;
```

Insister sur le côté "combine" de la solution, c'est plus un casse-tête amusant que de l'informatique. C'est l'occasion de dire aux étudiants que si la réponse avait été "non", il aurait été très difficile de le prouver : comment prouver qu'un programme ne peut pas faire quelque chose ?

4.

```
#include<stdio.h>

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

  a=3; b=5; c=7;

  printf("a vaut %i\n", a);
  printf("b vaut %i\n", b);
  printf("c vaut %i\n", c);

   t=a; //On sauvegarde la valeur de a
   a=b;
// ... On pourrait continuer comme ca
   b=c;
   c=t;
```

```
  printf("a vaut %i\n", a);
  printf("b vaut %i\n", b);
  printf("c vaut %i\n", c);
}
```

Pour les meilleurs étudiants, il est naturel de se demander “peut-on faire cela sans variable de sauvergarde ?”. La réponse est oui puisqu'on peut échanger deux variables quelconques. Et en échangeant deux par deux, on peut engendrer n'importe quelle permutation (les transpositions engendrent le groupe des permutations comme disent les algébristes). Il d'autres solutions, comme celle-ci trouvée par un étudiant lors de la première séance de TD en octobre 2007 :

```
a=a+b+c ; c=a-(b+c) ; b=a-(c+b) ; a=a-(b+c) ;
```

## TD 2 : Les boucles

Exercice 5. Écrire un programme qui affiche un tableau pour convertir les degrés Fahrenheit en degrés Celsius. Par exemple, votre programme affichera sur la colonne de gauche les nombre de 0 à 100 (de 5 en 5) correspondant aux degrés Fahrenheit, et sur la colonne de droite les valeur correspondantes en Celcius.

Rappel : $c = 5 \times (f - 32)/9$

Corrigé :

Pas de difficulté. Bien noter le $\backslash t$ qui permet d'aligner sans se fatiguer et les formats d'affichage.

```
#include<stdio.h>

main(){
  float celc, fahr;

  fahr = 0.0;

  printf("Fahrenheit\tCelcius\n");

  while (fahr <=100){
    celc = 5*(fahr-32)/9;
    // Plusieurs formats d'affichage
    // En chosir un.
    // printf("%f\t%f\n", fahr, celc); // \t tabule
       printf("%8.2f\t%8.2f\n", fahr, celc);
     fahr = fahr + 5;
  }
}
```

Exercice 6. 1. Tracer l'exécution du programme suivant :

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

  const int N=10;
  int i;
  int som;
  float harmo;

  harmo = 0.0; som = 0; i = 1; /* initialisation des variables */
```

```
  //Point d'observation 1

  while (i<=N){
    som = som + i;
    harmo = harmo + 1/(float)i;
    i++;
    // Point d'observation 2
  }
  // Point d'observation 3

  printf("La somme des %d premiers entiers est : %d\n", N, som);
  printf("La somme des %d premiers termes de la"
         " serie harmonique vaut : %f\n", N,harmo);
}
```

Corrigé :

| Point d'observation | i | som | harmo |
|---|---|---|---|
| 1 | 1 | 0 | 0.0 |
| 2 | 2 | 1 | 1,0 |
| 2 | 3 | 3 | 1,5 |
| 2 | 4 | 6 | 1,833 |
| 2 | 5 | 10 | 2,083 |
| 2 | 6 | 15 | 2,283 |
| 2 | 7 | 21 | 2,450 |
| 2 | 8 | 28 | 2,593 |
| 2 | 9 | 36 | 2,718 |
| 2 | 10 | 45 | 2,829 |
| 2 | 11 | 55 | 2,929 |
| 3 | 11 | 55 | 2,929 |

Pour les matheux, un joli exercice : montrer que les sommes partielles de la série harmonique ne sont jamais des nombres entiers. Si des étudiants s'ennuient, ils peuvent y réfléchir et le programmer pour voir. Évidemment, un jour ou l'autre l'ordinateur trouvera une somme partielle "entière" à son sens (si on n'y prend pas garde).

Exercice 7. On rappelle que la suite de Fibonacci est définie par : $f_1 = f_2 = 1$ et $\forall n \geq 3$, $f_n = f_{n-1} + f_{n-2}$. On pose pour tout $n \geq 2$, $u_n = f_{n+1}f_{n-1} - f_n^2$. Écrire un programme qui calcule $u_n$ et affiche sa valeur pour $n = 2, \ldots, a$ où $a$ est un nombre entré par l'utilisateur.

Corrigé :

Normalement, les suites à double récurrence ont été vues en cours. Donc, programmer Fibonacci est facile. Mais il y a une petite difficulté supplémentaire :

on doit garder 3 termes en mémoire et non pas 2.

```
#include<stdio.h>

main(){
  int s, p, pp, i, n, u;

  printf("Entrez un entier svp : ");
  scanf("%i", &n);


  pp=1;
  p=1;
  s=2;

  i=2;

  while(i <= n){
    printf("u%i vaut %i\n", i, u);
    //printf("f%i vaut %i\n", i, p); //utile pour des tests

    pp = p; //f(n-1)
    p = s;  //f(n)
    s = p + pp; //f(n+1)
    u = pp*s - p*p; //u(n)

    i++;
  }
}
```

On remarque $u_n = (-1)^n$. Ca n'a aucun intérêt pour nous de le démontrer en TD, mais si des étudiants le demande, on a toujours l'air moins bête en sachant répondre :

On peut vérifier que $1 \times 0 - 1 = (-1)^1$ puis par récurrence : $f_{n+2}f_n - f_{n+1}^2 = (f_{n+1}+f_n)f_n - f_{n+1}^2 = f_{n+1}(f_n - f_{n+1}) + f_n^2 = -f_{n+1}f_{n-1} + f_n^2 = (-1)^{n+1}$. Ca peut aussi se montrer en utilisant la relation bien connue :

$$f_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

On notera que le programme est étonnamment stable pour les grands nombres. Même quand on dépasse de loin les capacité de calcul de Fibonacci, le calcul de $u_n$ reste correct. Parce que la formule de $u_n = (-1)^n$ reste vrai avec des nombres de Fibonacci calculés modulo un grand entier.

Exercice 8. 1. Écrire un programme qui demande 2 entiers $x, y$ à l'utilisateur et qui affiche en retour un rectangle de $x$ lignes et $y$ colonnes. Par exemple, si l'utilisateur choisit $x = 5, y = 3$ le programme devra afficher :

```
xxx
xxx
xxx
xxx
xxx
```

2. modifier le programme précédent. Cette fois, un seul nombre $x$ est demandé et le programme affiche un triangle de $x$ lignes comme suit :

```
x
xx
xxx
xxxx
xxxxx
```

3. Dernière modification, le programme doit afficher le triangle ci-dessous pour $x = 5$ :

```
    x
   xxx
  xxxxx
 xxxxxxx
xxxxxxxxx
```

Corrigé :

```
#include<stdio.h>

main(){
  int x, y, i, j;

  printf("Entrez x svp : ");
  scanf("%i", &x);
  printf("Entrez  y svp : ");
  scanf("%i", &y);

  for(i=1; i<=x; i++){
    for(j=1; j<=y; j++){
```

```
    putchar('x');
    }
    putchar('\n');
  }

  putchar('\n');

  for(i=1; i<=x; i++){
    for(j=1; j<=x-i; j++){
    putchar(' ');
    }
    for(j=1; j<=2*i-1; j++){
    putchar('x');
    }
    putchar('\n');
  }
}
```

Exercice 9. 1. Programmer l'algorithme d'Euclide vu en cours (pour calculer le PGCD). Le programme doit en plus afficher le nombre de soustractions nécessaires pour parvenir au résultat.

2. On propose l'algorithme suivant pour le calcul du pgcd de $a$ et $b$.

$r_0 = |a|$
$r_1 = |b|$
Pour tout $n$, si $r_{n-1} \neq 0$, alors on définit $r_n$ par :
$r_n = |r_{n-2} - r_{n-1}|$

On peut montrer que le dernier terme non nul de la suite $(r_n)$ est le PGCD de $a$ et $b$.

Programmer cet algorithme. Le programme doit en plus afficher le nombre de soustractions nécessaires pour parvenir au résultat.

3. Comparer les deux algorithmes en terme de nombre d'itérations dans la boucle.

Corrigé :

On remarque que l'algorithme par soustraction est beaucoup moins performant. C'est l'occasion de dire qu'il y a de bons et de mauvais algorithmes.

Le if n'a pas encore été vu, donc on ne peut pas calculer la valeur absolu (ou alors avec des ruses affreuses avec while, qu'il ne faut surtout pas montrer si on veut que les étudiants fassent bien la différence en boucle et test). Le plus simple est d'utiliser la bibliothèque math.h. (mon intention en TD, c'est de dire d'utiliser abs, puis que les étudiants voient planter la compilation, puis enfin de dire d'inclure <math.h>).

Une remarque sur la programmation d'abs : peut-on programmer abs avec seulement les opération +, *, -, / ? La réponse est non, car abs n'est pas dérivable en 0, et tout ce qu'on fera de partout défini avec +, *, -, / le sera. C'est intéressant de voir que de l'analyse montre qu'un langage informatique restreint a des limites.

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

main(){
  int a, b, s, i, sa, sb;

  printf("Entrez a svp : ");
  scanf("%i", &sa);
  printf("Entrez  b svp : ");
  scanf("%i", &sb);
```

```
  /*ALGORITHME PAR SOUSTRACTIONS*/

  a=abs(sa); b=abs(sb);
  i=0;

  while(b!=0){
    s=a; //s comme sauvegarde
    a=b;
    b=abs(s-b);
    i++;
 }

  printf("Le PGCD de a et b vaut %i.\n", a);
  printf("L'algorithme par soustractions a necessite %i soustractions.\n", i);


  /* EUCLIDE */

  a=abs(sa); b=abs(sb);
  i=0;

  while(b!=0){
    s=a; //s comme sauvegarde
    a=b;
    b=s%b;
    i++;
 }

  printf("Le PGCD de a et b vaut %i.\n", a);
  printf("L'algorithme d'Euclide a necessite %i divisions.\n", i);
}
```

Exercice 10. Écrire un programme permettant d'afficher les nombres de 1 à $n$, ainsi que leur carré, et leur cube (soit trois colonnes). L'entier $n$ est au choix de l'utilisateur. Écrire ce programme trois fois : avec `while`, avec `do ... while` et avec `for`.

Exercice 11. Écrire un programme permettant d'afficher le triangle de Pascal. Le nombre de lignes est au choix de l'utilisateur.

```
          1
      1       1
```

```
          1     2     1
      1     3     3     1
   1     4     6     4     1
1     5     10    10    5     1
...
```

Rappel : chaque case du triangle reçoit la somme des valeurs dans les deux cases au dessus d'elle. Mais cette propriété ne dit pas comment programmer. On conseille de suivre la méthode ci-dessous :

On sait d'après le cours de mathématiques que la $k^{\text{e}}$ case ($k = 0, \ldots, n$) de la $n^{\text{e}}$ ligne reçoit la valeur $C_n^k = n!/((n-k)!k!)$. On en déduit immédiatement :

$$C_n^0 = 1, C_n^k = \frac{n-k+1}{k} C_n^{k-1}$$

Attention, il faudra bien veiller au fait que si la division $n!/((n-k)!k!)$ tombe toujours juste (puisque $C_n^k$ est un nombre entier), il n'en va pas de même de la division $(n-k+1)/k$.

Corrigé : L'exercice est trop difficile. Intéressant pour les meilleurs étudiants. Ne pas le corriger en séance. Au besoin, je mettrai un corrigé en ligne.

Il y a plusieurs difficultés, la moindre n'étant pas de "centrer" les lignes. Cet exercice est l'occasion de montrer qu'il faut faire attention en "traduisant" en langage C des formules mathématiques. C'est le programme le plus compliqué depuis le début, il faut peut-être commencer à dire aux étudiants de mettre des commentaires.

```
/* Triangle de Pascal */

#include<stdio.h>

main(){
  int n, i, c, j;

  printf("Bonjour, jusqu'a quelle ligne voulez-vous aller : ");
  scanf("%i", &n);

  for (i=0; i<=n; i++){  //pour chaque ligne

    // On ajoute des espaces pour centrer
    for(j=1; j<=(n-i)/2; j++) printf("      ");
    for(j=1; j<= 3*((i+n)%2); j++) printf(" ");

    // On calcule chaque case
    c=1; printf("%6i", c);
```

```
    for(j=1; j<=i; j++){
      c = c*(i-j+1)/j; //Correct
    //c = (i-j+1)/j*c; //La, c'est faux !
      printf("%6i", c);
    }

    // On va a la ligne !!
    printf("\n");
  }
}
```

Exercice 12. Pour chacun des deux programmes suivants, le recopier en l'indentant, le tracer, et dire ce qui s'affiche à l'execution. Expliquer la différence essentielle entre les deux programmes, et dire lequel des deux correspond le mieux aux spécifications annoncées dans les commentaires.

```
/* Programme 1 */
/* Affichage des carre et des cube des petits entiers */
#include<stdio.h>

main(){
int a, carre, cube;
printf("Nombre\t Carre\t cube\n");
//Point d'observation 1
for(a=1; a<=5; a++){
carre = a*a;
cube = carre*a;
printf("%i\t", a);
printf("%i\t", carre);
printf("%i\n", cube);
//Point d'observation 2
}
}
```

```
/* Programme 2 */
/* Affichage des carre et des cube des petits entiers */
#include<stdio.h>

main(){
int a, carre, cube;
printf("Nombre\t Carre\t cube\n");
//Point d'observation 1
for(a=1; a<=5; a++)
```

```
carre = a*a;
cube = carre*a;
printf("%i\t", a);
printf("%i\t", carre);
printf("%i\n", cube);
//Point d'observation 2
}
```

Exercice 13. QCM (traditionnellement, l'examen de langage C comporte toujours un petit QCM)

1. gcc est :
   a. Un identificateur du langage C
   b. Une commande qu'on tape dans la fenêtre de commande
   c. Un mot-clef du langage C
   d. Un opérateur du langage C

2. while est :
   a. Un identificateur du langage C
   b. Une commande qu'on tape dans la fenêtre de commande
   c. Un mot-clef du langage C
   d. Un opérateur du langage C

3. Dans le programme `main(){int a ; a=3%5 ;}`, % est :
   a. Un identificateur du langage C
   b. Une commande qu'on tape dans la fenêtre de commande
   c. Un mot-clef du langage C
   d. Un opérateur du langage C

4. Dans le programme `main(){int a ; a=3%5 ;}`, a est :
   a. Un identificateur du langage C
   b. Une commande qu'on tape dans la fenêtre de commande
   c. Un mot-clef du langage C
   d. Un opérateur du langage C

5. = permet en langage C de :
   a. réaliser une affectation
   b. tester une égalité
   c. comparer deux identificateurs
   d. convertir un int en float

6. En typographie le caractère & s'appelle :
   a. La luette
   b. L'espagnolette
   c. L'esperluette
   d. L'arpette

Corrigé :

Rappel : en C, il 6 “unités lexicales de base”, encore appelés lexèmes : les identificateurs, les mot-clefs, les séparateurs, les constantes, les constantes de type chaîne, les opérateurs.

1. b
2. c
3. d
4. a
5. a
6. c. Mais qui veut gagner des millions ?

## TD 3 : Les tests

Exercice 14. Écrire un programme qui demande deux entiers $a$ et $b$ à l'utilisateur et indique en retour si $b$ divise ou non $a$.

Exercice 15. Écrire un programme qui lit une note $n$ sur 20, puis affiche la mention correspondante : ajourné si $n < 10$, passable si $10 \leq n < 12$, assez bien si $12 \leq n < 14$, bien si $14 \leq n < 16$, et très bien sinon.

Corrigé : Deux solutions : ou bien tester avec des if emboités, ou bien avec des formules booléennes. La première solution est plus efficace, surtout si on connait les cas les plus fréquents, à mettre en premier. La deuxième solution est plus proches des spécifications, et peut-être plus lisible (question de goût).

```
#include<stdio.h>

main(){
  float note;

  printf("Votre note svp : ");
  scanf("%f", &note);

  // Avec des if emboite's :
  if (note < 0) printf ("Pas de notes negatives\n"); else
    if (note<10) printf ("Ajourne\n"); else
      if (note<12) printf("passable\n"); else
        if (note < 14) printf("AB\n"); else
          if (note < 16) printf("B\n"); else
            if (note <= 20) printf ("TB\n"); else
              printf("Pas de notes au dessus de 20\n");

  printf("\n\n");

  // Avec des formules booleennes :
  if (note < 0) printf ("Pas de notes negatives\n");
  if (note >= 0 && note < 10) printf ("Ajourne\n");
  if (note >= 10 && note < 12) printf("passable\n");
  if (note >= 12 && note < 14) printf("AB\n");
  if (note >= 14 && note < 16) printf("B\n");
  if (note >= 16 && note <= 20) printf("TB\n");
  if (note > 20) printf("Pas de notes au dessus de 20\n");
}
```

Exercice 16. Écrire un programme qui demande 3 nombres à l'utilisateur, et les affiche en retour, mais triés par ordre croissant. Idem pour 4 nombres.

Corrigé : Multitude de solutions. On peut demander aux étudiants de dénombrer les résultats possibles dans le cas général : $n!$ facons d'ordonner $n$ entiers, soit 6 pour ce qui nous concerne. Conséquence intéressante : comme chaque if nous permet de discerner deux cas, $k$ usages de if nous mènent à $2^k$ possibilités. Donc, à moins d'utiliser des ruses mathématiques plus ou moins scabreuses, des case, ou je ne sais quoi, il faut au moins 3 if pour trier trois entiers ($2^2 < 3! \leq 2^3$). De même, il faut au moins 5 if pour trier quatre entiers ($2^4 < 4! \leq 2^5$). On doit pouvoir trier 5 entiers avec seulement 7 if ($2^6 < 6! \leq 2^7$), mais là ça devient de la combinatoire ...

Noter que sans affectations, chaque solution doit donner lieu à un printf "spécialisé", et donc, il semble qu'on soit obligé d'utiliser $n!$ if.

```
#include<stdio.h>

main(){
  int a, b, c, d;
  int aa, bb, cc, dd;
  int t;

  printf("Entrez un entier a : ");
  scanf("%i", &a);
  printf("Entrez un entier b : ");
  scanf("%i", &b);
  printf("Entrez un entier c : ");
  scanf("%i", &c);
  printf("Entrez un entier d : ");
  scanf("%i", &d);


  aa=a; bb=b; cc=c; dd=d; //On sauvegarde les valeurs pour faire plusieurs tests

  // Premiere methode : lourd
  if (a <= b && b <= c) printf("%i, %i, %i\n", a, b, c);
  if (a <= c && c <= b) printf("%i, %i, %i\n", a, c, b);
  if (b <= a && a <= c) printf("%i, %i, %i\n", b, a, c);
  if (b <= c && c <= a) printf("%i, %i, %i\n", b, c, a);
  if (c <= a && a <= b) printf("%i, %i, %i\n", c, a, b);
  if (c <= b && b <= a) printf("%i, %i, %i\n", c, b, a);


  a=aa; b=bb; c=cc; d=dd; //On remet à jour les valeurs
```

```
  //Une solution mixte
  if (a>b) {t=a; a=b; b=t;} //Maintenant, a<=b.
  if (c<=a) printf("%i, %i, %i\n", c, a, b); else
    if (c<=b) printf("%i, %i, %i\n", a, c, b); else
      printf("%i, %i, %i\n", a, b, c);


  a=aa; b=bb; c=cc; d=dd; //On remet à jour les valeurs

  //Une solution qui trie vraiment
  if (a>b) {t=a; a=b; b=t;} //Maintenant, a<=b. restent 3 cas
  if (a>c) {t=a; a=c; c=t;} //Maintenant, a<=c. restent 2 cas
  if (b>c) {t=b; b=c; c=t;} //Maintenant, b<=c.

  //Maintenant a<=b<=c
  printf("%i, %i, %i\n", a, b, c);


  a=aa; b=bb; c=cc; d=dd; //On remet à jour les valeurs

  //Une solution pour 4 nombres avec seulement 5 if
  if (a>b) {t=a; a=b; b=t;} //Maintenant, a<=b. restent 12 cas
  if (c>d) {t=c; c=d; d=t;} //Maintenant, c<=d. restent 6 cas
  if (b>d) {t=b; b=d; d=t;} //Maintenant, b<=d. restent 3 cas
  if (a>c) {t=a; a=c; c=t;} //Maintenant, a<=c. restent 2 cas
  if (b>c) {t=b; b=c; c=t;} //Maintenant, b<=c.

  //Maintenant a<=b<=c<=d
  printf("%i, %i, %i, %i \n", a, b, c, d);

  //Peut-on ecrire une solution aussi simple pour 5 nombres ? Mystere...
}
```

Exercice 17. On décrit le jeu des allumettes : au départ, il y a un tas de 50 allumettes, (ou tout autre objet : cailloux, jetons, ...). Chacun à son tour, les deux joueurs ôtent obligatoirement entre 1 et 6 allumettes. Celui qui ôte la dernière allumette gagne.

1. Préférez-vous commencer ou jouer en deuxième ? Justifiez votre choix par une stratégie gagnante. Conseil : commencer par raisonner avec un petit nombre d'allumettes, puis généraliser.

2. Écrire un programme qui joue au jeu des allumettes contre l'utilisateur.

Corrigé :

1. On peut démontrer qu'une position est gagnante pour celui qui a la main si et seulement si le nombre d'allumettes n'est pas divisible par 7.

Preuve : supposons que cela soit faux, et considérons le plus entier $n$ qui contredirait ce qu'on veut montrer. Si $1 \leq n \leq 7$, c'est évident. Supposons $n \geq 8$.

Si $n$ est divisible par 7 tandis que la position est gagnante, il y a une contradiction, car tout coup jouable mène à un entier non divisible par 7 qui par hypothèse de récurrence est une position gagnante. C'est bien une contradiction : d'une position gagnante il doit y avoir au moins un coup donnant à mon adversaire une position perdante.

Si $n$ n'est pas divisible par 7 tandis que la position est perdante, il y a une contradicition, car enlever $n \mod 7$ allumette mène à un entier divisible par 7 qui par hypothèse de récurrence est une position perdante. C'est bien une contradiction : d'une position perdante, tout coup doit donner à mon adversaire une position gagnante.

2.

```
#include<stdio.h>

#define ORDI   0
#define HUMAIN 1

main(){
  int coup, pos, joueur;

  printf("Combien d'allumettes : ");
  scanf("%i", &pos);
  printf("Qui commence (%i pour l'ordi, %i pour vous) : ", ORDI, HUMAIN);
  scanf("%i", &joueur);

  while(pos!=0){
    if (joueur == ORDI){
      coup = pos%7;
      if (coup == 0) coup = 1;
      printf("Il y a %i allumettes, j'en enleve %i.\n", pos, coup);
    }
    else{
      printf("Il y a %i allumettes. Votre coup svp : ", pos);
      scanf("%i", &coup);
    }
    pos = pos - coup;
```

```
    joueur = !joueur;
  }
  if (joueur == ORDI) printf("Vous avez gagne !\n");
  else printf("Vous avez perdu...\n");

  return 0;
}
```

Exercice 18. Exercice de révision : jusqu'où irez vous ? À chacune des questions suivantes, on demande d'écrire un programme. On conseille à chaque fois d'enregistrer, de compiler et d'exécuter le programme, de toujours travailler sur le même fichier et de faire "grossir" le programme de questions en questions.

Si vous voulez vous auto-évaluer : faites ce test en salle machine. Comptez deux points pour les questions 1 et 2, trois pour les suivantes. Ca vous fera une note assez approximative sur 19.

1. Écrire un programme qui affiche "Bonjour".

2. Faire en sorte que le programme demande à l'utilisateur d'entrer un nombre de son choix. Le programme doit ensuite afficher : "Le nombre que vous avez demandé est : *valeur*".

3. Le programme doit ensuite afficher un message indiquant si le nombre est oui ou non divisible par 7.

4. Le programme doit maintenant afficher la liste des diviseurs du nombre initialement entré. Par exemple, si l'utilisateur entre initialement le nombre 12, le programme doit écrire : "Les diviseurs de 12 sont 1, 2, 3, 4, 6 et 12".

5. On rappelle qu'un nombre est premier s'il n'est divisible que par 1 et par lui-même. Par convention, 1 n'est pas considéré comme un nombre premier. Le programme doit indiquer si le nombre entré par l'utilisateur est ou non premier.

6. Le programme doit maintenant afficher la liste de tous les nombres premiers inférieurs au nombre entré par l'utilisateur.

7. Le programme doit mainteant afficher seulement les nombre premier jumeaux. Deux nombres premiers sont dits *jumeaux* si leur différence est égale à 2 (ou à $-2$).

Remarque : à l'heure actuelle, personne ne sait s'il existe un nombre fini ou infini de nombres premiers jumeaux.

Corrigé : Pas besoin de faire cet exercice en séance. Si un groupe est

en avance, le chargé peut faire l'autoévaluation pour attendre les autres groupes.

## TD 4 : Les caractères

Exercice 19. Afficher tous les caractères du jeu de caractères du langage C, avec leur code ASCII.

Corrigé :

```
#include<stdio.h>

main(){
  char c;

  c=0;

  do {
    putchar(c);
    printf("   %i\n", c);
    c++;
  } while(c!=0);

}
```

Noter que les char sont signés ou non selon l'implémentation. Ils vont de -128 à 127, ou de 0 à 255. Mettre un `unsigned` en cas de désir d'éviter les char négatifs (mais pourquoi vouloir les éviter ? ?). Quoiqu'il en soit, en partant de zéro et en incrémentant, on retournera à zéro exactement quand on aura tout parcouru.

Bien noter donc la condition d'arrêt `c!=0`. D'autres choix de conditions apparemment raisonnables ne marchent pas. Par exemple, `c<=127` (ou `c<=255` en unsigned) est vrai pour tout char, et le programme bouclera. Quant à `c<127` (ou `c<255` en unsigned), ça ne bouclera pas, mais on ratera le caractère 127 (ou 255 en unsigned).

Beaucoup de caractères ne sont pas affichables. Le numéro 7 produit normalement un "bip" : moyen le plus simple de faire du bruit en langage C.

Exercice 20. 1. Écrire un programme qui lit le flot de caractères tapés au clavier, et l'affiche à l'écran. Le programme doit s'interrompre quand l'utilisateur tape le caractère de fin de fichier. Ce caractère, dont le numéro est noté conventionnellement EOF est accessible à l'utilisateur depuis le clavier en tapant CTRL D (ça peut dépendre du système, sous Windows essayer CTRL Z).

2. À l'aide d'une redirection de la sortie, utiliser le programme de la question précédente pour saisir un fichier appelé texte1.

3. À l'aide d'une redirection de l'entrée, utiliser le programme de la première question pour recopier le fichier de la question précédente dans un fichier appelé copie1.

Corrigé :

```
#include<stdio.h>

main(){
  char c;

  c = getchar();
  while (c!=EOF){
    putchar(c);
    c = getchar();
  }
}
```

Normalement, les fonctions qui gèrent les fichiers comme `fprintf, fscanf` ne sont pas au programme du premier semestre. Chaque chargé de TD est libre de les enseigner à ses étudiants, mais cela demande d'avoir pas mal de temps. Pour écrire dans un fichier dans le contexte de ce TD, il est plus simple d'utiliser le mécanisme des redirections, vu en cours.

Exemple : le programme `printf(‘‘bonjour’’);` affiche bonjour dans la console. Mais, comme dit le proverbe, sous UNIX, "tout est fichier". En fait, le programme affiche bonjour dans un fichier très particuler, appelé la sortie standard. Et par défaut, la sortie standard s'affiche à l'écran (qui est aussi une sorte de fichier). Si au lieu de taper ./prog on tape ./prog>toto, la sortie sera redirigée vers le fichier appelé toto. Si on tape ./prog<tata, les scanf seront lus non plus au clavier, mais à partir du contenu du fichier tata.

Exercice 21. Écrire un programme qui demande un caractère à l'utilisateur, et qui affiche en retour "le caractère est un chiffre", "le caractère est une lettre minuscule", ou "le caractère est une lettre majuscule", selon la valeur du caractère. Si le caractère n'appartient à aucune des catégories ci-dessus, le programme doit afficher le message "autre sorte de caractère".

Exercice 22. Le but de cet exercice est d'écrire un programme qui lit l'entrée et établit quelques statistiques simples. Tester ce programme au clavier, puis en redirigeant un fichier vers l'entrée, par exemple le fichier ballade.text disponible sur le site web du langage C.

1. Nombre de caractères.

2. Nombre de lettres majuscules.

3. Nombre de chiffres.

4. Nombre de lignes.

5. Longueur de la plus longue ligne.

Corrigé :

On s'autorise à utiliser certaines particularités du code ASCII : les lettres majuscules sont consécutives, les lettres minuscules idem, et les chiffres aussi. Le résultat est un programme théoriquement non portable, et les puristes préferront utiliser des fonctions de la bibliothèque comme `isletter`, `isdigit`, etc.

Pas de difficulté notable, sauf la définition précise de la longueur d'une ligne. Le caractère de fin de ligne en fait-il partie ? Question de peu d'intérêt : adopter une convention et s'y tenir.

```
#include<stdio.h>

main(){
  char c;

  int nb_char, nb_maj, nb_chiffre, nb_ligne, ligne_long;
  int i; //Ou on en est dans une ligne

  nb_char = 0;
  nb_maj = 0;
  nb_chiffre = 0;
  nb_ligne = 0;
  ligne_long = 0;


  c = getchar();
  while (c!= EOF){

    nb_char++;
    i++; // On avance dans la ligne
    if ('A' <= c && c<= 'Z') nb_maj++;
    if ('0' <= c && c<= '9') nb_chiffre++;
    if (c=='\n') {
      nb_ligne++;
      if (i>ligne_long) ligne_long = i-1;
      i=0; // On repart a zero dans la nouvelle ligne
    }
```

```
    c = getchar();

  }

  //Le nombre de lignes est faux sauf en cas de derniere ligne vide
  if (i>ligne_long) ligne_long=i-1;
  if (i>0) nb_ligne++;

  //Affichage des resultats


  printf("nb_char : %i\n",  nb_char);
  printf("nb_maj : %i\n",  nb_maj);
  printf("nb_chiffre : %i\n",  nb_chiffre);
  printf("nb_ligne : %i\n",  nb_ligne);
  printf("ligne_long : %i\n",  ligne_long);
}
```

Exercice 23. Écrire un programme de code secret de votre choix. Ce programme lit un flot de caractères en entrée, et retourne un flot codé sur la sortie. Bien penser à écrire aussi le programme qui décode. Appliquer ce programme au codage et décodage de fichiers textes (par exemple ballade.text).

Piste à suivre la plus simple : décaler toute les lettres de 1, ou d'un entier $k$ à fixer. Plus subtil : décaler la première lettre de 1, la deuxième de 5, la suivante de 1, puis de 5, etc ... La conception de codes secrets efficaces est une science à part entière : la cryptographie.

Corrigé :

Pas de difficulté spéciale, sinon de bien rester dans le jeu de caractères affichables. Solution parmi d'autres pour décaler assez proprement une lettre minuscule c de 10 :

```
c = 'a' + ((c-'a') + 10)%26;
```

Notez qu'on obtient le célèbre code de l'avocat ('a' vaut 'k') ... Ca ne marche que pour des textes composés de lettres minuscules seulement. Pour coder des textes plus généraux, il y a des problèmes un peu embêtants : les retours à la lignes notamment.

# TD 5 : Les fonctions

Exercice 24. Écrire un programme qui demande à un commercial son chiffre d'affaires, qui le lise et calcule le montant de sa prime de rendement. La prime est égale à 1% de la partie du chiffre d'affaires comprise entre 20 000€ et 50 000€, plus 2% de ce qui dépasse 50 000€.

Le programme comportera deux fonctions : la fonction `main()` et une fonction de prototype `float prime(float)` qui calcule la prime selon la formule ci-dessus.

Exercice 25. 1.

Écrire une fonction de prototype `int f (int, int, int)` qui prend en entrée trois entiers et qui affiche les informations suivantes :

- Tous les entiers compris entre le plus grand et le plus petit ;
- Les trois entiers, mais triés par ordre croissant ;
- Somme du plus grand et du plus petit ;
- Nombre réel égal à $a/b$ où $a$ est plus grand, et $b$ le plus petit.

La fonction doit renvoyé l'écart entre le plus grand et le plus petit.

2.

Écrire une fonction main() qui demande à l'utilisateur trois entiers et affiche toutes les informations ci-dessus, et qui affiche aussi l'écart entre le plus grand entier et le plus petit.

Exercice 26. 1. Tracer le programme suivant :

```
#include<stdio.h>

// Suites de Syracuse

int suivant(int u){
  int parite;

  parite = u%2;

  if (parite == 0) u = u/2;
  else u = 3*u+1;

  //Point d'observation 1

  return u;
}

main(){
```

```
  int n;

  n=5;

  while (n!=1){
    printf("%i ", n);
    n=suivant(n);
    //Point d'observation 2
  }

  return 0;
}
```

2. Que donnerait le traçage en remplaçant la ligne $n = 5$ par $n = 27$ ?

Corrigé : 1. Conformément à ce qui a été vu en cours, tracer, c'est donner à chaque passage au point d'observation la valeur de toutes les variables. À l'intérieur d'une fonction, comme on n'a pas accès aux variables locales des autres fonctions, celles-ci ont pour valeur "indéterminée". Avant le nom de chaque variable, on précise la fonction, pour distinguer les éventuelles homonymies.

| Pt obs | main n | suivant u | suivant parite |
|---|---|---|---|
| 1 | indéterminée | 16 | 1 |
| 2 | 16 | indéterminée | indéterminée |
| 1 | indéterminée | 8 | 0 |
| 2 | 8 | indéterminée | indéterminée |
| 1 | indéterminée | 4 | 0 |
| 2 | 4 | indéterminée | indéterminée |
| 1 | indéterminée | 2 | 0 |
| 2 | 2 | indéterminée | indéterminée |

2. Avec $n = 27$, l'exécution du programme montre que le traçage aurait été plutôt long mais sans qu'il y ait bouclage. Pour toute valeur de départ, atteint-on la valeur 1 ? Autrement dit, le programme peut-il boucler ? À ce jour, la question est ouverte ! Cette question, surgie à l'université de Syracuse dans les années 1950, a fait perdre tellement de temps à tellement de chercheurs qu'on a cru que c'était un coup des Soviétiques pour saboter la recherche américaine. Quelle époque c'était.

# TD 6 : Systèmes de numération et autres amusements mathématiques

Exercice 27. On cherche à éprouver la célèbre formule :

$$e^x = \lim_{n\to+\infty} \sum_{i=0}^{n} \frac{x^i}{i!}$$

1. Programmer une fonction qui calcule la factorielle d'un nombre entier. Prototype de la fonction : `int fact(int)`.

2. Programmer une fonction qui calcule une valeur approchée de $\lim_{n\to+\infty} \sum_{i=0}^{n} x^i/i!$. On additionnera des termes de la forme $x^i/i!$ tant que $x^i/i! \geq 0{,}000\ 001$. Prototype de la fonction à programmer : `float devexp(float)`.

3. Comparer le résulat obtenu avec celui de la fonction exponentielle de la bibliothèque `math.h` du langage C. Pour accéder à cette fonction, `#include<math.h>` et selon les systèmes, option -lm à la compilation. Prototype de la fonction : `float exp(float)`

Exercice 28. 1. Écrire une fonction de prototype `void conv(int)` convertit un entier en base 2. Cette fonction affiche les bits de l'entier $n$ passé en argument.

2. Écrire une fonction de prototype `void conv(int n, int B)` convertit un entier $n$ en base $B$. Cette fonction affiche les chiffres en base $B$ de l'entier $n$ passé en argument. La fonction doit fonctionner pour les bases 2 à 16 au moins.

Corrigé :

L'application directe de l'algorithme vu en cours a un petit défaut : les chiffres sont affichés à l'envers. Ça n'est pas grave : l'énoncé ne demande pas de les afficher l'endroit. Résoudre ce problème en toute généralité nécessite d'utiliser un algorithme inefficace ou des tableaux. Ci-dessous, une combine pour afficher à l'endroit en base 2.

```
#include<stdio.h>

void conv2(int n){

  while (n!=0){
    printf("%i", n%2);
    n=n/2;
  }
  printf("\n");
```

```
}

void convB(int n, int B){

  int c; char affc;

  while (n!=0){
    c=n%B;
    if (c<=9) affc='0'+c; else affc='A'+c-10;
    printf("%c", affc);
    n=n/B;
  }
  printf("\n");
}


void convAlendroit(int n){
  int res, d;

  res=0;
  d=1;

  while (n!=0){
    res = res + n%2*d;
    d=10*d;
    n=n/2;
  }
  printf("%i", res);
  printf("\n");
}



main(){
  convB(255, 16);
}
```

EXERCICE 29. 1. Écrire une fonction de prototype `void inv(int)` qui calcule et affiche les chiffres en base 10 de l'inverse d'un entier $n$ passé en argument.

2. Vérifier la variante suivante du petit théorème de Fermat : le développement en base 10 de l'inverse d'un entier premier $p$ différent de 2 et 5 est de la forme

$0, \overline{a_1 a_2 \dots a_k}$ où $k$ est un diviseur de $p-1$.

Indication : on pourra utiliser le fait qu'en base 10, les inverses des entiers qui ne sont ni divisibles par 2 ni par 5 sont de la forme $0, \overline{a_1 a_2 \dots a_k}$. La période peut être détectée lors de l'exécution de l'algorithme par le retour de 1 dans la suite des restes.

3. Le petit théorème de Fermat tel qu'énoncé ci-dessus permet-il de tester si un nombre est premier ?

Corrigé :

```
#include<stdio.h>
#define B 10

// Affiche les chiffres de l'inverse
void affinv(int n){
  int r; int q;

  r=1;

  do{
    q=(B*r) / n;
    r=(B*r) % n;
    printf("%i", q);
  }
  while (r!=1);

  printf("\n");
}


//Calcule la longueur de la période
int periode(int n){
  int r; int q; int i;

  r=1; i=0;

  do{
    q=(B*r) / n;
    r=(B*r) % n;
    //printf("%i", q);
    i++;
  }
  while (r!=1);
  //printf("\n");
```

```
  return i;
}



main(){
  int n, d;
  int prem, ferm;

  for(n=2; n<10000; n++){
    if (n%2!=0 && n%5!=0){
      //On teste si n est premier
      d=2;
      while(d<=n/2 && n%d != 0) d++;
      prem = (n%d != 0);

      ferm = (n-1) % periode(n)  == 0;

      if (ferm && prem) printf("%i confirme le petit theoreme de Fermat\n", n);
      if (!ferm && prem) printf("%i infirme le petit theoreme de Fermat\n", n);
      if (ferm && !prem) printf("%i infirme la reciproque du petit theoreme de Ferma

    }

  }

}
```

1. Voir ci-dessus.

2. Voir ci-dessus.

3. L'exécution du programme montre que la réciproque du théorème de Fermat admet des contre-exemples, comme 9, 561 ou 1729 (même s'il y en a assez peu). On ne peut donc pas utiliser le théorème de Fermat pour tester si un nombre est premier. Changer de base permet d'éliminer certain de ces contre-exemples, mais pas tous. Les nombres de Carmichaël sont par définition les nombres contre-exemple en toute base. Les plus petits nombre de Carmichaël sont 561, 1105 et 1729. Noter que 1729 est bizarre à plus d'un titre : c'est le plus petit entier s'exprimant de deux manière différentes comme somme de deux cubes : $1729 = 10^3 + 9^3 = 12^3 + 1^3$.

# TD 7 : Tableaux et chaînes de caractères

Dans tous les exercices sur les tableaux, il est recommandé de définir à l'aide du préprocesseur une constante `MAX`, en plaçant la ligne `#define MAX 10` par exemple eu début du programme.

Exercice 30. 1. Écrire une fonction de prototype `int mini(int[])` qui renvoit le plus petit élément du tableau passé en argument.

2. Écrire une fonction de prototype `int indice_mini(int[])` qui renvoit l'indice d'un plus petit élément du tableau.

3. Écrire une fonction de prototype `int trier(int a[], int b[])` qui recopie les éléments du tableau $a$ dans le tableau $b$, mais en les triant par ordre croissant.

Corrigé :

Exercice 31. 1. Écrire un programme qui stocke les entiers de 0 à 100 dans un tableau. Pour tout $0 \leq i \leq 100$, on devra trouver l'entier $i$ à la case $i$.

2. Modifier le programme pour qu'à la case $i$ on trouve non plus $i$ mais la somme des carrés des entiers de 0 à $i$.

Exercice 32. Dans cet exercice, on définira avec le préprocesseur une constante MAX. Les tableaux qu'on déclarera auront MAX cases.

1. Écrire une fonction de prototype `void permutte(int tab[], int i, int j)` qui permutte les éléments $i$ et $j$ du tableau tab.

2. Écrire une fonction de prototype `int indiceM(int tab[])` qui renvoit le plus petit indice $i$ vérifiant `tab[i] < tab[i-1]`. Si un tel indice n'existe pas, la fonction renverra MAX.

3. Écrire une fonction `void tri(int tab[])` qui trie le tableau tab par ordre croissant. On utilisera l'algorithme suivant : tant que i=indiceM est différent de MAX, permutter les cases $i$ et $i-1$ du tableau.

Corrigé :

```
#include<stdio.h>
#define MAX 4


void afftab(int tab[]){
```

```
  int i;

  for(i=0; i<MAX; i++)
    printf("%i %i\n", i, tab[i]);
}


void permutte(int tab[], int i, int j){

  int t;

  t = tab[i];
  tab[i] = tab[j];
  tab[j] = t;
}


int indiceM(int tab[]){

  int i;

  i=0;

  while(i<MAX && tab[i] >= tab[i-1])
    i++;

  return i;
}


void tri(int tab[]){
  int i;

  while ((i=indiceM(tab)) < MAX)
    permutte(tab, i-1, i);
}

main(){
  int t[MAX];

  t[0] = 100;
  t[1] = 90;
  t[2] = 80;
```

```
  t[3] = 10;

  //printf("Indice : %i\n", indiceM(t));
  //permutte(t, 3, 6);

  tri(t);

  afftab(t);
}
```

Exercice 33. 1. Écrire une fonction qui donne la longueur d'une chaîne de caractères passée en argument.

2. Écrire une fonction de prototype `int compare (char a[], char b[])` qui renvoit vrai si la chaîne a est placée avant la chaîne b dans l'ordre alphabétique. On suppose que $a$ et $b$ sont des chaîne de lettres minuscules.

3. Reprendre la question précédentes en supposant que $a$ et $b$ sont des chaînes de lettres quelconques (majuscules ou minuscule)

Corrigé :

Exercice 34.

```
#include<stdio.h>

#define MAX 10

int f(int n, int t[]){
int l, c;

l=0;
while(n!=0){
//Point d'observation 1
c=n%2;
n=n/2;
t[l]=c;
l++;
//Point d'observation 2
}
```

```
return l;
}

main(){
int tab[MAX];
int i, l;

l=f(12, tab);

for (i=l-1; i>=0; i--)
printf("%i", tab[i]);
printf("\n");
}
```

1. Recopier le programme ci-dessus en l'indentant.

2. Tracer le programme ci-dessus. Attention : ne pas s'occuper du tableau `t`, ni des variables défines dans `main()`. A chaque point d'observation, indiquer uniquement les valeurs des variables c, n, et l.

3. Expliquer ce que fait la fonction `f` en fonction de `n`.

4. Que vaut la fonction `f` en fonction de `n` ?

5. Qu'affiche le programme ?

6. Qu'affiche le programme si à la place de `f(12, tab)` on lui demande `f(x)` où `x` est un entier préalablement déclaré et initialisée ?

EXERCICE 35. 1. Écrire une fonction de prototype `void conv2(int n, int nb2[], int *l)` qui stocke dans le tableau nb2 les bits de l'entier $n$ exprimé en base 2, et qui stoke dans `*l` le nombre de chiffres en base 2 de $n$ auquel on ôte 1. Par exemple, si $n = 6$, qui s'écrit 110 en base 2, l'appel à conv2 aura pour effet nb2[0] = 1, nb2[1] = 1, nb2[2] = 0 et *l=2 (car il y a 3 chiffres dans 110).

2. Écrire une fonction de prototype `int conv10(int nb2[], int l)` qui renvoie l'entier dont les chiffres en base 2 au nombre de $l + 1$ sont stockés dans nb2. On utilisera la factorisation de Horner pour $B = 2$ :

$$\sum_{i=0}^{l} a_i B^i = a_0 + B(a_1 + B(\dots))$$

Indication : Soit $n = a_0 2^0 + a_1 2^1 + \cdots + a_l 2^l$ un entier exprimé en base 2. Utiliser la factorisation de Horner revient à calculer les termes de la suite

définie par : $u_{l+1} = 0$ et pour tout $0 \leq k \leq l$, $u_k = 2u_{k+1} + a_k$. On a alors $u_0 = n$.

3. Écrire une fonction de prototype `int troisP(int n)` qui renvoit $3^n$. La fonction devra utiliser la méthode suivante. Appeler `conv2(n, nb2, &l)` pour calculer les chiffre en base 2 de $n$ puis utiliser la formule suivante :

$$3^n = 3^{a_0+2(a_1+2(\dots))} = 3^{a_0}\left(3^{a_1+2(\dots)}\right)^2$$

4. Essayer de réécrire la fonction précédente sans appel à conv2. C'est-à-dire qu'il faut calculer les chiffres en base 2 au fur et à mesure du calcul de $3^n$.

5. Estimer le nombre de multiplications nécessaires pour calculer $3^n$ par la méthode précédente. Comparer ce nombre au nombre de multiplications utilisées par la méthode naïve.

Corrigé :

Exercice 36. Écrire une fonction de prototype `convlettre(int n, char lettre[])` qui écrit dans la chaîne lettre le nombre $n$ en écrit en français et en toute lettres. La fonction devra supporter les nombres jusqu'à 999.

Corrigé :

```
#include <stdio.h>
char * nb_0_19[20]={"zéro","un","deux","trois","quatre","cinq",
                    "six","sept","huit","neuf","dix","onze",
                    "douze","treize","quatorze","quinze","seize",
                    "dix-sept","dix-huit","dix-neuf"};
char * dizaine[]={"","","vingt","trente","quarante","cinquante",
                  "soixante","","quatre-vingt"};

void affiche_0_19(int n)
{ printf("%s",nb_0_19[n]);
}

void affiche_0_99(int n)
{ int d,u,v;
  d=n/10;
  switch(d)
  { case 0:
    case 1:affiche_0_19(n);
```

```
            break;
    case 2: case 3: case 4: case 5: case 6:printf("%s",dizaine[d]);
                                           u=n%10;
                                           if (u!=0)
                                           { printf("%s",(u==1)?" et ":"-");
                                             affiche_0_19(u);
                                           }
                                           break;
    case 7: case 8:case 9:printf("%s",dizaine[(d/2)*2]); // 7->6 ; 8 et 9 -> 8
                          v=n%20;
                          if (d==8 && v==0) printf("s");
                          else
                          { if(d==7 && v==11) printf(" et ");
                            else printf("-");
                            affiche_0_19(v);
                          }
                          break;
  }
}

void affiche_0_999(n)
{ int c,r;
  c=n/100;
  r=n%100;
  if (c>0)
  { if (c>1) printf("%s ",nb_0_19[c]);
    printf("cent");
    if (c>1 && r==0) printf("s ");
    else printf(" ");
  }
  if (c==0 || r!=0) affiche_0_99(r);
}

main()
{ int i;
  for(i=0;i<300;i++)
  { affiche_0_999(i);
    printf("   ");
  }
  putchar('\n');
}
```

EXERCICE 37. 1. Écrire une fonction de prototype `int indice(int t[],`

`n)` qui renvoie un indice $i$ tel que `t[i] = n` si un tel entier existe. Sinon, la fonction renvoie -1.

2. On suppose maintenant que le tableau t est trié par ordre croissant. Il va alors de soi que la recherche d'un indice tel que `t[i] = n` doit pouvoir être accélérée. Implémenter l'algorithme par recherche dichotomique : poser inf=0 et sup=MAX où MAX est le plus grand indice d'un élément du tableau. Tant que $t[\text{sup}] > t[\text{inf}]$, calculer $m = (\text{sup} + \text{inf})/2$, comparer $t[m]$ à $n$ et remettre à jour inf et sup en fonction du résultat. À tout moment, on doit avoir $t[\text{inf}] \leq n \leq t[\text{sup}]$.

Corrigé :

## Renseignements utiles

**Commandes UNIX utiles :**

- `ls` : affiche la liste des fichiers du répertoire courant.
- `cd` : changer le répertoire courant.
- `mkdir` : crée un répertoire
- `pwd` : affiche le nom du répertoire courant
- `rm` : efface un fichier
- `gcc` : compile un programme en langage C
- `clear` : efface l'écran
- `man` : affiche une documentation sur une commande UNIX

**Autres choses utiles dans la fenêtre de commande :**

- `.` : désigne le répertoire courant
- `..` : désigne le répertoire juste au dessus du répertoire courant dans l'arborescence des repertoires
- `TAB` : permet de completer automatiquement le nom d'un fichier ou d'une commande
- ↑ : permet de rappeler la dernière commande tapée
- `CTRL-C` : permet d'interrompre l'exécution d'un programme

**Important :** comment sauvegarder son travail sur une clef USB ? Ca n'est pas si simple, car linux gère mal les périphériques en général. Une solution qui a fonctionné en salle C701 :

- Entrer la clef dans un port USB de l'ordinateur
- Taper dans une fenêtre de commande `cd /mnt`
- Taper `ls` et essayer de deviner dans les répertoires qui s'affichent quel est celui correspondant à la clef USB (ça varie selon les situations, ça peut être `removable`, ou `UDISK`, ou autre chose encore)
- Taper `cd removable` (ou `UDISK`, ou autre chose selon l'étape précédente)
- Taper `ls` et vérifier que la liste des fichiers qui s'affichent correspond bien à ceux devant se trouver originelement dans la clef
- Taper `cd`
- À l'aide la commande `cp`, copier tous les fichiers que vous souhaitez sur votre clef. Par exemple `cp nometudiant/prog.c /mnt/removable`
- Désinstaller la clef en tapant `umount /mnt/removable`
- Enlever la clef du port USB

Il est aussi possible de sauvegarder son travail sur disquette, ou de l'envoyer par courrier électronique (à soi même par exemple).

**Comment travailler chez soi le langage C**

Si vous avez un ordinateur "exotique" (Atari ?) ou très ancien, il faut cherche par vous même s'il y a un compilateur C disponible. Ca risque d'être compliqué. Si vous avez un ordinateur Mac pas trop ancien, le langage C doit être installé dessus par défaut. Si vous avez un PC tournant sous LINUX,

idem. Si vous avez un PC tournant seulement sous WINDOWS, vous avez deux solutions.

- Ou bien intaller LINUX. Cette option est réservé à ceux qui sont prêts à y passer du temps ou à ceux pouvant se faire aider par un connaisseur. En effet l'opération peut être compliquée, risquée et entrainer des pertes de données. Notez que LINUX est gratuit. Une recherche sur le web donne de nombreuses possibilités pour le télécharger, etc.
- Ou bien installer un compilateur C gratuit fonctionnant sous WINDOWS. Par exemple lcc-win32. Un lien vers le site est disponible sur l'EPI langage C, rubrique Cours Premier Semestre, Liens Intéressants.

**Sur Emacs**

Emacs est un "super éditeur". Il dispose de modes d'édition spécialisés pour de nombreux langages, comme le C, mais aussi le LISP, TEX, LATEX, etc. Heureusement, par bien des aspects, Emacs fonctionne comme un éditeur de texte classique. On ouvre des fichiers, on les modifie, on les enregistre. Les fonctions classiques de couper/coller, recherche d'une chaîne, etc sont accessibles à partir des menus déroulants. Un petit détail : quand on veut créer un nouveau document, il faut aller dans "fichier", puis "ouvrir fichier". Ensuite, il faut bien penser à tout de suite saisir le nom du fichier dans le "mini-buffer", c'est-à- dire dans la ligne tout en bas de la fenêtre. Si on oublie cette denière étape, Emacs laisse des explications précédées de " ;" au début du fichier.

Quelques raccourcis clavier bien pratiques :

- TAB : indente en mode langage C
- CTRL X S : enregistrer
- CTRL X F : ouvrir un fichier
- CTRL X W : enregistrer sous
- CTRL X 2 : dédouble la fenêtre
- CTRL X 0 : ferme la fenêtre courante
- CTRL Espace : defini le début d'une région (= d'une sélection dans les traitement de texte)
- $\leftarrow, \rightarrow, \downarrow, \uparrow$ : permettent de définir la région selectionnée à partir du CTRL-Espace.
- CTRL W : couper
- ESC W : copier
- CTRL Y : coller
- CTRL K : couper la ligne courante
- CTRL S : rechercher dans le buffer courant
- ESC % : rechercher-remplacer
- CTRL L : replace la ligne courante au milieu

## Systèmes de numération

On appelle système de numération tout procédé permettant d'associer à chaque nombre une suite de caractères. Le système de numération en usage aujourd'hui est la numération dite de position en base 10, mais d'autres systèmes ont été utilisés et le sont encore : les chiffres romains par exemple. Il se trouve que dans l'état actuel des technologies, le système le plus commode pour les ordinateurs est la représentation de position en base 2, encore appelée représentation binaire.

### Développement d'un entier en base $B$

Soit $B \geq 2$ un entier. On appelle *chiffres en base $B$* les entiers compris entre 0 et $B-1$. Donc, une fois la base fixée, il y a un nombre fini de chiffres, et en théorie, il est toujours possible de représenter chaque chiffre par un symbole.

**Théorème 1** *Soit $B \geq 2$ un entier. Soit $n > 0$ un entier. Alors il existe une unique suite d'entiers naturels $(a_k, a_{k-1}, \ldots, a_1, a_0)$ telle que :*

- $a_k \neq 0$
- *Pour tout $i$, $a_i$ est un chiffre en base $B$.*
- $n = a_0 B^0 + a_1 B^1 + a_2 B^2 + \cdots + a_k B^k$.

Cette suite s'appelle le *développement en base $B$* de $n$. Les $a_i$ s'appellent les *chiffres* de $n$. On écrit $n = a_k a_{k-1} \ldots a_1 a_0$. Dans l'usage courant en informatique, les bases les plus utilisées sont la base 10 et la base 2. Un entier écrit en base deux est dit sous forme *binaire*, en base 10 sous forme *décimale*. On appelle *bits* les chiffres d'un entier en base 2 (de l'anglais 'binary digit'). Les bases 8 et 16 sont aussi utilisées par les informaticiens : on parle de nombres donnés en *octal* et en *hexadécimal*. En hexadécimal, le nombre de chiffres depasse 10. On utilise les lettres A, B, C, D, E, F pour noter les chiffres après 9.

### Trouver les chiffres d'un nombre

Pour trouver les chiffres d'un nombre $n$ dans une base $B$, il suffit d'être capable de calculer quelques divisions Euclidiennes :

$q_0 = n$
Pour tout $i \geq 0$ on définit $a_i$ et $q_{i+1}$ par :
$q_i = q_{i+1} B + a_i$, avec $0 \leq a_i < B$

**Théorème 2** *À partir d'un rang $k+1$, la suite $(a_i)$ devient nulle. Et on a, en base $B$ : $n = a_k a_{k-1} \ldots a_0$.*

PREUVE — On montre d'abord que pour tout $i \in \mathbb{N}$ on a $n = q_i B^i + a_0 B^0 + \cdots + a_{i-1} B^{i-1}$. Pour $i = 0$, c'est vrai car $q_0 = n$. Si la propriété est vraie

pour $i \geq 0$, alors $n = q_i B^i + a_0 B^0 + \cdots + a_{i-1} B^{i-1} = (q_{i+1} B + a_i) B^i + a_0 B^0 + \cdots + a_{i-1} B^{i-1} = q_{i+1} B^{i+1} + a_0 B^0 + \cdots + a_i B^i$. La propriété est bien vraie pour $i+1$, et de là pour tout $i \in \mathbb{N}$.

Comme $(B^i)_{i \in \mathbb{N}}$ tend vers l'infini, les $q_i$ doivent tous être nuls à partir d'un certain rang, et les $a_i$ le sont donc aussi. La relation ci-dessus montre bien $n = a_k a_{k-1} \dots a_0$. □

### Représentation des nombres réels

La représentation des nombres réels pose des problèmes théoriques et pratiques. En effet, le mathématicien Cantor a montré à la fin du XIX$^{\text{e}}$ siècle qu'étant donné un ensemble fini de symboles $A$, il est impossible d'associer sans ambiguïté à chaque nombre réel une suite finie d'éléments de $A$. Toutefois, il existe de nombreux systèmes permettant d'associer à chaque réel une suite éventuellement infinie de symboles.

Afin de ne pas alourdir le cours, on se restreint au cas des réels de $[0, 1[$. Cela a pour avantage que tous les nombres s'écriront $0, \dots$. Et cela est suffisant, car tout nombre réel est la somme d'un entier relatif et d'un réel de $[0, 1[$. Soit $(a_n)_{n \in \mathbb{N}^*}$ une suite de chiffres en base $B$, et soit $x \in [0, 1[$ un nombre réel. On pose :

$$x_n = a_1 B^{-1} + a_2 B^{-2} + \cdots + a_n B^{-n}$$

On dit que $(a_n)_{n \in \mathbb{N}}$ est une représentation de $x$ si $x = \lim_{n \to \infty} x_n$. On écrit alors $x = 0, a_1 a_2 a_3 \dots$. Il faut noter que pour n'importe quel choix de la suite $(a_n)_{n \in \mathbb{N}^*}$, la suite $(x_n)_{n \in \mathbb{N}^*}$ converge.

Dans le cas où la suite $(a_n)_n \in \mathbb{N}^*$ est périodique, de période $p$ à partir du terme $a_k$, on s'autorise la notation suivante :

$$x = 0, a_1 a_2 a_3 \dots a_{k-1} \overline{a_k a_{k+1} \dots a_{k+p-1}}$$

Attention, certains nombres réels peuvent avoir deux représentations différentes en base $B$. Par exemple, en base 10, on a $1/2 = 0{,}5 = 0,499999 \cdots = 0, 4\overline{9}$.

### Algorithme pour trouver le développement des rationnels

L'algorithme suivant, qui n'est autre que celui enseigné aux enfants à l'école primaire, permet de trouver un développement en base $B$ pour tout nombre rationnel de $[0, 1[$. On se donne en entrée deux entiers naturels $u$ et $v$, avec $u < v$. On définit les suites $(r_k)_{k \in \mathbb{N}}$ et $(q_k)_{k \in \mathbb{N}^*}$ :

$r_0 = u$
Pour tout $k > 0$ on définit $r_k$ et $q_k$ par :
$B r_{k-1} = v q_k + r_k$, avec $0 \leq r_k < v$

On remarque que pour tout $k$, $r_k < v$. Donc, $0 \leq q_k < B$, autrement dit les entiers $q_k$ peuvent être vus comme des chiffres en base $B$. Le théorème suivant montre qu'en fait, les suites définies ci-dessus permettent d'obtenir le développement en base $b$ de $u/v$ :

**Théorème 3** $u/v = 0, q_1 q_2 q_3 \ldots$

PREUVE — On montre d'abord pour tout $k \geq 0$ :

$$\frac{r_k}{B^k v} = \frac{u}{v} - \left(\frac{q_1}{B^1} + \cdots + \frac{q_k}{B^k}\right)$$

Pour $k = 0$, c'est évident car $r_0 = u$. Si $k \geq 1$, supposons la formule vraie pour $k-1$. Puisque $r_k = B r_{k-1} - v q_k$, on a alors :

$$\frac{r_k}{B^k v} = \frac{r_{k-1}}{B^{k-1} v} - \frac{q_k}{B^k} = \frac{u}{v} - \left(\frac{q_1}{B^1} + \cdots + \frac{q_k}{B^k}\right)$$

par hypothèse de recurrence.

Comme $r_k < v$, on a donc :

$$\frac{u}{v} - \left(\frac{q_1}{B^1} + \cdots + \frac{q_k}{B^k}\right) < \frac{1}{B^k}$$

ce qui montre bien que :

$$\lim_{k \to +\infty} \frac{q_1}{B^1} + \cdots + \frac{q_k}{B^k} = \frac{u}{v}$$

□

Encore une fois, l'algorithme nous permet de démontrer un résultat théorique, qui a pour conséquence, entre autres, l'existence de nombres irrationnels :

**Théorème 4** *Un nombre réel est rationnel si et seulement si il admet un développement périodique en base $B$.*