Exercices de langage C

6 mars 2007

Exercice 1

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^{e} case ($k = 0, \ldots, n$) de la n^{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

Il y a plusieurs difficultés, la moindre n'étant pas de "centrer" les lignes.

```
/* 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 2

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 3

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. Il y a $n!$ 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 4

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 5

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 Fermat\n", n)

    }

  }

}
```

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 : $1729 = 10^3 + 9^3 = 12^3 + 1^3$.