Programmation en C/C++

Série 15
Écriture d'algorithmes

(Série non finalisée)

 Important !

 Difficile !

Objectifs

Écrire quelques algorithmes en langage C++ traitant de problématiques classiques.

EXERCICES

Exercice 1

Ecrire un programme C++ qui convertit un nombre entier en base 10 (compris entre 0

Tableau de correspondances

Chiffres romains	I	V	X	L	C	D	M
Nombres décimaux	1	5	10	50	100	500	1000

Exemples

1637 : MDCXXXVII

3009 : MMMIX

499 : CDXCIX

Règles de conversion

- Au plus, trois répétitions consécutives de M, C, X et I. Par conséquence, avec cette notation, le plus grand chiffre romain est 3999 puisque MMMM n'existe pas.

  Exemples

  - 4 : IV et non IIII
  - 40 : XL et non XXXX
  - 400 : CD et non CCCC

- Au plus, une seule occurrence D, L et V.
- Un seul C peut préfixer (placé avant) un M ou un D.
  - 800 : DCCC et non CCM
- Les chiffres après M ou D représentent une valeur qui ne peut être supérieure à 99.
- Un seul X peut préfixer un C ou un L.
  - 30 : XXX et non XXL
- Les chiffres après C ou L représentent une valeur qui ne peut être supérieure à 9.
- Un seul I peut préfixer un X ou un V
  - 7 : VII et non IIIX
- Toute lettre placée à droite d'une autre de valeur supérieure ou égale s'ajoute à celle-ci.
  - CXXX => 100 + 10+10+10 donc 130

- Toute lettre placée à la gauche d'une autre lettre plus forte qu'elle, indique que le nombre qui lui correspond doit être retranché du nombre relatif à la seconde
    - CCM : 1000 – 100 – 100 => 800
- L'écart de poids ne peut être que d'une unité
    - 99 : XCIX et non IC

Corrigé

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

int Convert(int NB, char * ChiffreRomain) {
    char* SymbolesRomains[]={"M","CM","D","CD","C","XC","L","XL","X","IX",
        "V","IV","I"};
    int ValeursDecimales[]={1000,900,500,400,100,90,50,40,10,9,5,4,1};
    int i=0;
    if (NB<=0 || NB >3999)
        return 0;
    strcpy(ChiffreRomain,"");

    while (NB>0) {
        while (NB >= ValeursDecimales[i]) {
            NB-=ValeursDecimales[i];
            strcat(ChiffreRomain,SymbolesRomains[i]);
        }
        i+=1;
    }
    return 1;
}

int main(void) {
    int Nombre;
    int Retour;
    char ChiffreRomain[1024];
```

```
    do {
        printf("Entrer un nombre entier compris entre 0 et 3999 ");
        scanf("%d",&Nombre);
        Retour=Convert(Nombre,ChiffreRomain);
        if (Retour==1)
            printf("En chiffres romains : %s\n",ChiffreRomain);
        else
            printf("Sortie : il faut respecter la plage de valeurs !");
    } while(Retour==1);

    return 0;
}
```

Exercice 2 à trouver

Ecrire un programme C++ qui convertit un nombre en chiffres romains en un nombre entier en base 10.

Corrigé

Exercice 3

Énoncé

Saisir un entier long à plusieurs chiffres. Le programme appelle une fonction qui retourne son nombre de chiffres. Pensez aux logarithmes base 10 (*log10*) et à la librairie *Math.h* qui permet justement d'utiliser les logarithmes.

Corrigé

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

long NbChiffres (long N)
{
return (int) (1 + log10(N));
}

int main()
```

```
{
long Nombre;
printf("saisir un nombre : \n");
scanf("%ld",&Nombre);
printf("Nombre de chiffres : %ld \n",NbChiffres(Nombre));
return 0;
}
```

Exercice 4

Énoncé

Saisir un entier long à plusieurs chiffres. Le programme affiche le nombre de chiffres puis si cet entier est un palindrome ; à l'aide de deux fonctions distinctes NbChiffres et Palindrome.

Exemple : 4334, 5 et 383 sont trois palindromes.

Corrigé

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

long NbChiffres (long N)
{
return (int) (1 + log10(N));
}

int Palindrome(long n) {
  // on compare le nombre n et son nombre miroir nbis que l'on crée
  long a = n;
  long nbis = 0;

  while (a != 0) {
    nbis = nbis * 10 + a % 10;
    a /= 10;
  }
```

```
  return nbis == n;
}

int main()
{
long Nombre, longueur;
printf("saisir un nombre : \n");
scanf("%ld",&Nombre);

printf("Nombre de chiffres : %ld \n",NbChiffres(Nombre));
if (Palindrome (Nombre) )
   printf ("Ce nombre est un palindrome\n");
else
   printf ("Ce nombre n'est pas un palindrome\n");

return 0;
}
```

Exercice 5 Tri par sélection

Énoncé

Le tri par sélection consiste à parcourir un tableau à trier afin de trouver l'élément le plus petit. Cet élément est alors permuté avec le premier élément du tableau. Le premier élément du tableau est donc à sa place. Il reste ensuite à trier ensuite le **sous-tableau non trié** de la même façon. Il s'agit d'un algorithme relativement lent.

10-24-5-7-12

5-24-10-7-12

5-7-10-24-12

5-7-10-24-12

5-7-10-12-24

En vert : partie non triée dans laquelle on cherche le plus petit élément.

Exemple d'itérations

15, 10, 23, 2, 8, 9, 14, 16,

2, 10, 23, 15, 8, 9, 14, 16,

2, 8, 23, 15, 10, 9, 14, 16,

2, 8, 9, 15, 10, 23, 14, 16,

2, 8, 9, 10, 15, 23, 14, 16,

2, 8, 9, 10, 14, 23, 15, 16,

2, 8, 9, 10, 14, 15, 23, 16,

2, 8, 9, 10, 14, 15, 16, 23,

Corrigé

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

void Affiche (int Tableau[],int MaxTableau)
{
int i;
for(i = 0; i < MaxTableau; i++)
                printf("%d, ",Tableau[i]);
                printf("\n");
}

void tri_selection(int *tableau, int taille)
{
    int en_cours, plus_petit, j, temp;
for (en_cours = 0; en_cours < taille - 1; en_cours++)
{
        plus_petit = en_cours;
        for (j = en_cours ; j < taille; j++)
            if (tableau[j] < tableau[plus_petit])
                plus_petit = j;

          temp = tableau[en_cours];
```

```
        tableau[en_cours] = tableau[plus_petit];
        tableau[plus_petit] = temp;
          Affiche(tableau,8);
}
}

int main()
{
   int Tableau[8] = {15, 10, 23, 2, 8, 9, 14, 16};
   Affiche(Tableau,8);
   tri_selection (Tableau, 8);
   return 0;
}
```

Exercice 6 Tri par insertion

Énoncé

Le tri par insertion est le tri des joueurs de cartes…

Dans le tri par insertion, on parcourt le tableau restant à trier du début à la fin. On trouve le plus petit élément et on l'insère à sa place parmi les éléments triés. On recommence avec la partie du tableau qui n'est pas triée. Au moment où l'on traite le $n^{ème}$ élément qui n'est pas à sa place, tous les éléments qui le précèdent sont déjà triés. On insère donc le $n^{ème}$ élément dans les n-1 précédents déjà triés en les décalant. Il s'agit d'un algorithme lent.

→Écrire un programme qui réalise un tri par insertion et affiche à chaque itération le nouveau tableau trié.

Exemples d'itérations

15, 10, 23, 2, 8, 9, 14, 16,

10, 15, 23, 2, 8, 9, 14, 16,

10, 15, 23, 2, 8, 9, 14, 16,

2, 10, 15, 23, 8, 9, 14, 16,

2, 8, 10, 15, 23, 9, 14, 16,

2, 8, 9, 10, 15, 23, 14, 16,

2, 8, 9, 10, 14, 15, 23, 16,

2, 8, 9, 10, 14, 15, 16, 23,

Corrigé itératif

```
#include <stdio.h>
#include <stdlib.h>
void Affiche (int Tableau[],int MaxTableau)
{
int i;
for(i = 0; i < MaxTableau; i++)
                printf("%d, ",Tableau[i]);
                printf("\n");
}

void tri_insertion (int* t, int n)
{
   int i, j;
   double elementInsere;

   for (i = 1; i < n; i++) {
      /* Stockage de la valeur en i */
      elementInsere = t[i];
      /* Décale les éléments situés avant t[i] vers la droite
         jusqu'à trouver la position d'insertion */
            for (j = i; j > 0 && t[j - 1] > elementInsere; j--) {
               t[j] = t[j - 1];
            }
      /* Insertion de la valeur stockée à la place vacante */
      t[j] = elementInsere;
      Affiche(t,8);
  }
}

int main()
{
int Tableau[8] = {15, 10, 23, 2, 8, 9, 14, 16};
Affiche(Tableau,8);
tri_insertion (Tableau, 8);
return 0;
```

```
}
```

Exercice 7 Tri à bulles

Énoncé

L'algorithme de tri à bulles parcourt le tableau et compare les éléments 2 à 2. Lorsque deux éléments ne sont pas dans l'ordre, ils sont échangés.

Après un premier parcours complet du tableau, le plus grand élément est donc bien placé en fin de tableau. On procède de la même façon en parcourant à nouveau le tableau, en s'arrêtant à l'avant-dernier élément. Ainsi de suite et l'on arrête quand il n'y a pas eu d'échange. Il s'agit d'un algorithme relativement lent voire très lent si le nombre d'éléments à trier augmente fortement.

Exemple

Premier passage

10-24-5-7-12	10 et 24 sont déjà classés donc ils ne bougent pas
10-24-5-7-12	24 et 5 vont être permutés
10-5-24-7-12	
10-5-7-24-12	
10-5-7-12-24	

Second passage

10-5-7-12-24	5 et 10 sont permutés
5 10 7 12 24	10 et 7 sont permutés
5 7 10 12 24	Tout est classé

Corrigé

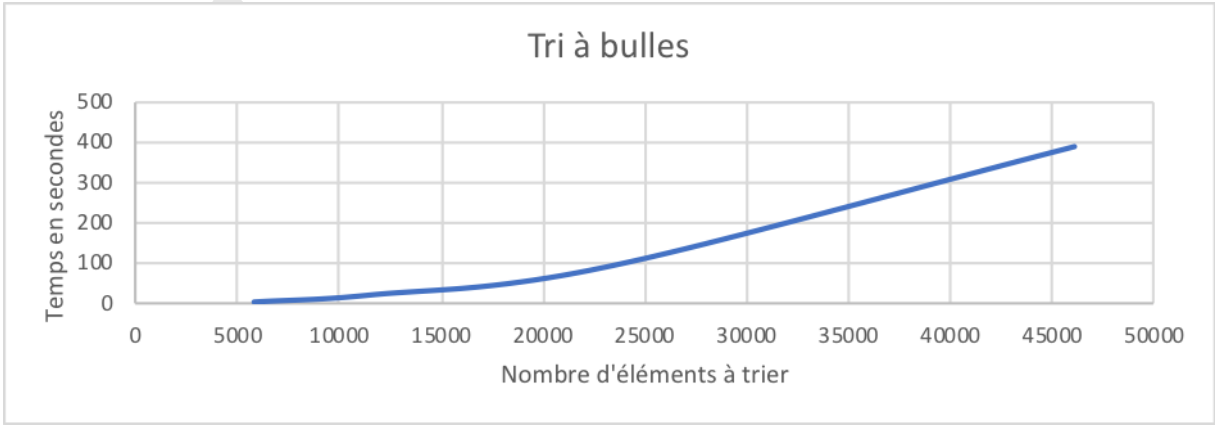
```
#include <stdio.h>
#include <stdlib.h>
void Affiche (int Tableau[],int MaxTableau)
{
int i;
for(i = 0; i < MaxTableau; i++)
                printf("%d, ",Tableau[i]);
                printf("\n");
}
```

```
int main()
{
   int K,L,I,J;
   int MaxTableau = 8;
   int Tableau[8] = {15, 10, 23, 2, 8, 9, 14, 16};
   Affiche(Tableau,8);
   for(I = MaxTableau - 2;I >= 0; I--) {
      for(J = 0; J <= I; J++) {
         if(Tableau[J + 1] < Tableau[J]) {
            int t = Tableau[J + 1];
            Tableau[J + 1] = Tableau[J];
            Tableau[J] = t;
         }
      }
   Affiche(Tableau,8);
   }
   return 0;
}
```

Remarque

Voici, pour cet algorithme, un tableau de valeurs avec des temps de calcul en fonction du nombre croissant d'éléments à trier.

Nb éléments	5760	11520	23040	46080
Temps (s)	5,6	22,9	90,87	385



Exercice 8 Tri rapide

Non traité ; hors programme

Énoncé

Le tri rapide consiste à choisir un élément du tableau et à générer 2 sous-tableaux T1 et T2 ; T1 contiendra toutes les valeurs strictement inférieures à l'élément et T2 contiendra toutes les valeurs strictement supérieures à l'élément. Le processus est ensuite réalisé sur les sous-tableaux. Comme son nom l'indique, il s'agit d'un algorithme rapide.

Exemple

125 4 56 356 7 12 8

56 est choisi ; on obtient : T1 : {4 7 12 8} l'élément 56 et T2 : {125 356}

On a donc T1 56 T2

soit : {4 7 12 8} 56 {125 356}

Travail sur T1

12 est choisi dans T1 ; on obtient : T11 : {4 -7-8} l'élément 12 T12 : {}

On a donc T11 12 T12

Soit {4 - 7 - 8} 12 {}

Travail sur T2

125 est choisi dans T2 ; on obtient : T21 : {} l'élément 125 T22 : {356}

On a donc T21 125 T22

Soit : {} 125 {356}

En résumé, après ce travail de niveau 2, on a donc :

T11 : {4 7 8} 12 T12 : {} 56 T21:{} 125 T22:{356}

Travail sur T11

Arrêt car T11 classé

Travail sur T12

Arrêt car T12 vide

Travail sur T21

Arrêt car T12 vide

Travail sur T22

Arrêt car T22 classé

Rien n'a changé, on a donc :
T11 : {4 7 8} 12 T12 : {} 56 T21:{} 125 T22:{356}
On concatène alors tous les tableaux pour obtenir le tableau final classé.
{4 7 8 12 56 125 356}

Corrigé

Exercice 9 Tri fusion

Non traité ; hors programme

Énoncé

L'algorithme de tri fusion est le suivant :

- Si le tableau ne possède qu'un seul élément, il est déjà trié.
    - Sinon, on sépare le tableau en deux parties à peu près égales.
- On trie récursivement les deux parties obtenues avec l'algorithme du tri fusion.
- On fusionne les deux tableaux triés en un tableau trié.

L'algorithme de tri fusion de deux listes en une repose sur le fait suivant : le plus petit élément de la liste finale à construire est soit le plus petit élément de la première liste, soit le plus petit élément de la deuxième liste.

La fusion triée de deux listes ne s'effectue que sur des listes elles-mêmes déjà triées.

Il s'agit d'un algorithme relativement rapide.

Corrigé

Exemple

Départ : T : {125 4 56 356 7 12 8}

Tri fusion de T : {125 4 56 356 7 12 8}
->T1 : {125 4 56 356} et T2 : {7 12 8}

Tri fusion de : T1
->T11 : {125 4} et T12 : {56 356}

Tri fusion de : T11
->T111 : {125} et T112: {4}

Tri fusion de T111
->T111 : {125} Fin

Tri fusion de T112
-> T112 : {4} Fin

Fusion T112 et T111
->{4,125}

Tri fusion de T12
-> T121 : {56} et T122 : {356}

Tri fusion de T121
-> T1211 : {56} Fin

Tri fusion de T122
-> T1221 : {356} Fin

Fusion T1211 et T1221
->{56,356}

Fusion de {4,125} et {56,356}
-> {4,56,125,356}

Tri fusion de T2
->T21 : {7, 12} et T22 : {8}

Tri fusion de T21
->T211 : {7} et T212 : {12}

Tri fusion de T211
-> T2111 : {7} Fin
Tri fusion de T212
-> T2121 : {12} Fin
Fusion de T2111 et T2121
->{7,12}

Tri fusion de T22
->T221 : {8} Fin

Fusion de ->{7,12} et {8}
->{7,8,12}

Fusion de {4,56,125,356} et {7,8,12}
->{4,7,8,12, 56,125,356}

Et continue par le tri fusion de T2
….

Exercice 10 Tri par tas

Non traité ; hors programme

Il s'agit d'un algorithme rapide.

Corrigé

Exercice 11 Les tours de Hanoï

Non traité ; hors programme

Énoncé

On dispose de 3 tiges et d'un certain nombre de disques de taille variable placés les uns au dessus des autres sur la tige A la plus à gauche.

L'objectif final est de placer tous les disques sur la tige C la plus à droite en respectant deux règles :

On ne peut déplacer qu'un disque à la fois sur la tige de son choix,

Un disque doit toujours être placé au dessus d'un disque de diamètre inférieur.

Position de départ avec 9 disques

Position d'arrivée avec 9 disques

Conseils

Utiliser un algorithme récursif. Il faut déplacer les (n-1) disques de la tige A sur la tige B qui joue le rôle de pivot puis recommencer vers la tige C. Le critère d'arrêt n=0 doit être bien placé pour avoir un nombre d'appel fini (en fait, 2^n).

Corrigé

```
#include <stdio.h>

void Hanoi(int nb,char A,char B,char C)
{
if(nb==1)
printf("deplacer l'element de la tour %c vers la tour %c\n",A,C);
else    {
Hanoi(nb-1,A,C,B);
printf("deplacer l'element de la tour %c vers la tour %c\n",A,C);
Hanoi(nb-1,B,A,C);
}
}

int main() {
char a='a',b='b',c='c';
int nb;
```

```
printf("Entrer le nombre d'element sur la tour de depart A : ");
scanf("%d",&nb);
printf("\n");

Hanoi(nb,a,b,c);
return 0;
}
```