Algorithmique et Programmation C++

TD 3

1. Miroir

```
/* TD3 - Exercice 1 - MIROIR       *\
\* Par Gael Even et Jeremie Osmont */


#include <iostream.h>

void miroir()
{
        char c;
        cin >> c;
        if(c!='.')
        {
                miroir();
                cout << c;
        }

}

int main()
{
        cout << "Entrez une phrase terminee par un point "
             << "pour obtenir son image miroir !\n";
        miroir();
        cout << "\n";
}
```

2. Fibonacci

```
/* TD3 - Exercice 2 - FIBONACCI    *\
\* Par Gael Even et Jeremie Osmont */


#include <iostream.h>

long fib(int n)
{
        if(n==0||n==1)
                return 1;
        else
                return fib(n-1)+fib(n-2);
}

int main()
{
        long x;
        cout << "Test de la fonction de Fibonacci : Entrez un entier
(superieur a 1) SVP\n> ";
        cin >> x;
        if (x<2) x=2;
        if (x>35) x=35;
        for (long j=0; j<=x; j++)
                cout << "fib(" << j << ") = " << fib(j) << "\n";
        cout << "\n";
}
```

3. PGCD

```
/* TD3 - Exercice 3 - PGCD         *\
\* Par Gael Even et Jeremie Osmont */


#include <iostream.h>

int min(int x,int y)
{
        // Donne le minimum de 2 nombres x et y
        if(x<=y) return x; else return y;
}

int max(int x,int y)
{
        // Donne le maximum de 2 nombres x et y
        if(x>=y) return x; else return y;
}

int pgcd(int x,int y)
{
        // Donne le PGCD par recursivite
        if(x==y) return x; else
                return pgcd(min(x,y),max(x,y)-min(x,y));
}

int main()
{
        cout << "Calcul du PGCD de 2 entiers longs positifs :\n";
        long a=1,b=1;
        cout << "> a="; cin >> a;
        cout << "> b="; cin >> b;
        if (a<=0) a=1;
        if (b<=0) b=1;
        cout << "\n> PGCD("<<a<<","<<b<<") = "<<pgcd(a,b);
        cout << "\n\n";
}
```

4. Palindrome

```
/* TD3 - Exercice 4 - PALINDROME   *\
\* Par Gael Even et Jeremie Osmont */


#include <iostream.h>

bool palindrome(char* pT,int i,int j)
{
        // Teste si la chaine est un palindrome par recursivite
        // Comme : "Esope reste ici et se repose" !
        if(j-i<=1)
                return true;
        else
                return (pT[i]==pT[j]) && palindrome(pT,i+1,j-1);
}

int main()
{
        char T[500];
        cout << "Bonjour, veuillez entrer une phrase (pas trop
longue) :\n> ";
        cin >> T;
        int i=0;
        while(T[i]!='\0') i++;
        if(palindrome(T,0,i-1))
                cout << "Bravo ! C'est un Palindrome !\n";
        else
                cout << "Dommage, ce n'est pas un Palindrome...\n";
}
```

5. & 6. Tri par insertion / Tri par fusion

```
/* TD3 - Exercices 5 & 6 - TRIS RECURSIFS *\
\* Par Jeremie Osmont            12/11/01 */


#include <iostream.h>
#include<math.h>
#include<stdlib.h>


// Jeu de fonctions de traitement de tableaux ////////////
long NbrAleat(long max)
{
        long a=rand();
        while (a>10000)
                a = a - 10000;
        return static_cast<long>(max*a/10000);
}

void TabAleat(long Tableau[],int taille, long max=99,long min=0)
{
        for (int i=0; i<taille; i++)
                Tableau[i]=NbrAleat(max-min)+min;
}

void TabDisp(long Tableau[], int taille, int min=0, int max=0)
{
        if (min<0 || min>taille) min=0;
        if (max<min || max>taille) max=taille-1;
        for (int a=min; a<=max;a++)
                cout<<Tableau[a]<<";";
        cout<<"\n";
}
//////////////////////////////////////////////////////////


// Variable espion ///////////////////////////////////////
long appels=0;
//////////////////////////////////////////////////////////


// Tri par insertion recursif ////////////////////////////
void inserer(long T[], int x)
{
        long aux;
        while (x>0 && T[x]<T[x-1])
        {
                aux=T[x];T[x]=T[x-1];T[x-1]=aux;
                x--;
        }
}

void triinsert(long T[], const int &n)
{
        appels++;
        if (n>0) {triinsert(T,n-1);inserer(T,n);}
}
//////////////////////////////////////////////////////////


// Tri par fusion recursif ///////////////////////////////
void fusion(long T[], const int &taille, const int &x, const int &y,
const int &z)
{
        long *C;
        C=new long[taille];
        int i=x, j=y+1, k=0;
        while (i<=y && j<=z)
        {
                if (T[i]<T[j]) {C[k]=T[i];i++;}
                else {C[k]=T[j]; j++;}
                k++;
        }
        for (int l=i; l<=y; l++)
                {C[k]=T[l]; k++;}
        for (int m=j; m<=z; m++)
                {C[k]=T[m]; k++;}
        k=0;
        for (int n=x; n<=z; n++)
                {T[n]=C[k]; k++;}
        delete[] C;
        return;
}

void trifusion(long T[], const int &taille, const int &m, const int
&n)
{
//cout << "Trifusion\n";
        if (m<n)
        {
                appels++;
                // Trie chaque moitie du tableau
                trifusion(T,taille,m,(m+n)/2);
                trifusion(T,taille,(m+n)/2+1,n);
                // Puis fusionne les deux parties
                fusion(T,taille,m,(m+n)/2,n);
        }
}
//////////////////////////////////////////////////////////


// Debut du programme ////////////////////////////////////
int main()
{
        int n=0;
        cout << "\nNombre de valeurs a trier : "; cin >> n;
        if (n<=0) {cout<<"Vu votre mauvaise volonte, au
revoir.\n";return 0;}

        long *T;
        if (!(T = new long[n]))
        {cout << "\nPAS ASSEZ DE MEMOIRE !\n"; return 0;}
        TabAleat(T,n);
        cout << "\nGeneration du tableau :\n";
        TabDisp(T,n,0,n-1);

        cout << "\nQuel type de tri recursif voulez-vous utiliser
?\n"
             << "-------------------------------------------------\n"
             << "  0 - Abandon\n"
             << "  1 - Tri par insertion\n"
             << "  2 - Tri par fusion\n"
             << "-------------------------------------------------\n";
        int tp=-1;
        do
        {
                cout << "> ";
                cin >> tp;
                if (tp>2 || tp<0) cout << "plaisantin !\n";
        }while(tp>2 || tp<0);
        switch(tp)
        {
                case 1 : triinsert(T,n);break;
                case 2 : trifusion(T,n,0,n-1);break;
        }

        if (tp>0)
        {
                cout << "\nTableau trie :\n";
                TabDisp(T,n,0,n-1);
                cout << "\nAppels recursifs : " << appels <<"\n";
        }
        cout << "\n";
        delete[] T;
}
//////////////////////////////////////////////////////////
```

7. Déterminant d'ordre N

```
/* TD3 - Exercice 7 - DETERMINANT MATRICIEL *\
\* Par Jeremie Osmont              22/11/01 */


#include <iostream.h>
#include <math.h>
#include <stdlib.h>
#include <iomanip.h>



// Changer N ici pour modifier la taille de la matrice :
// Desole c'est encore statique !
const int N=5;


// Generateur de nombres aleatoires /////////////////
long NbrAleat(long max)
{
        long a=rand();
        while (a>10000)
                a = a - 10000;
        return static_cast<long>(max*a/10000);
}
//////////////////////////////////////////////////


// Calcul du determinant version 'bourrin' //////////
// Copie pour chaque appel recursif une partie de la matrice
float det(float M[N][N], int c)
{
        float aux[N][N];
        float s=0;
        float signe=1;
        if(c==0)
                return M[0][0];
        for(int i=0;i<=c;i++)
        {
                for(int j=0;j<i;j++)
                        for(int k=0;k<c;k++)
                                aux[j][k]=M[j][k+1];
                for(j=i;j<c;j++)
                        for(int k=0;k<c;k++)
                                aux[j][k]=M[j+1][k+1];
                s+=signe*M[i][0]*det(aux,c-1);
                signe=-signe;
        }
        return s;
}
//////////////////////////////////////////////////////


// Calcul du determinant version 'fine' //////////////
// Met en oeuvre un descripteur qui indique les lignes valables
float xdet(float mat[N][N],int ligne[N], const int & taille)
{
        if (taille==1)
        {
                int j=0;
                while(ligne[j]==0) j++;
                return mat[j][N-1];
        }
        float s=0.0;
        float sig=1.0;
        for(int i=0; i<N; i++)
                if(ligne[i]==1)
                {
                        ligne[i]=0;
                        s+=sig*mat[i][N-
taille]*xdet(mat,ligne,taille-1);
                        sig=-sig;
                        ligne[i]=1;
                }
        return s;
}
//////////////////////////////////////////////////////


// Saisie d'un nombre ///////////////////////////////
int rd_entry(void)
{
        cout << "-----------------------------------------";
        cout << "\nEntrez un nombre :\n";
        int n=0;
        do
        {
                cout << "> ";
                cin >> n;
                if (n<2) cout << "Le nombre doit etre superieur a 1
!\n";
        }while(n<2);
        cout << "-----------------------------------------";
        return n;
}
//////////////////////////////////////////////////////


// Debut du programme ///////////////////////////////
int main()
{
        // Declaration statique de la matrice :
        float A[N][N];

        // Interface :
        cout << "\n-----------------------------------------";
        cout << "\n      CALCUL DE DETERMINANTS D'ORDRE "<< N;
        cout << "\n-----------------------------------------";
        cout << "\n    0 - Quitter";
        cout << "\n    1 - Utiliser la methode 'bourrin'";
        cout << "\n    2 - Utiliser la methode 'fine'";
        cout << "\n-----------------------------------------";
        cout << "\nEntrez votre choix :\n";
        int cx=-1;
        do
        {
                cout << "> ";
                cin >> cx;
                if (cx>2 || cx<0) cout << "Plaisantin !\n";
        }while(cx>2 || cx<0);

        // Fin du programme :
        if (cx==0)
        {
        cout << "-----------------------------------------\n\n";
        cout << "TSSS... Ca m'enerve ces gens qui lancent un ";
        cout << "programme et qui ne s'en servent meme pas...\n\n";
        return 0;  // Met fin au programme.
        }

        // Generation de la matrice aleatoire ;
        int max=rd_entry();
        for (int c=0; c<N; c++)
                for (int d=0; d<N; d++)
                        A[c][d]=static_cast<float>(NbrAleat(max)-
max/2+1);

        // Affichage de la matrice :
        cout<<"\nAffichage de la matrice "<< N << "x" << N << " :\n";
        for (int a=0;a<N;a++)
        {
                for (int b=0;b<N;b++) cout << setw(4) << A[a][b];
                cout<<"\n";
        }

        // Calcul du determinant :
        float d;
        int ligne[N];
        for(int i=0; i<N; i++) ligne[i]=1;
        if (cx==1) d=det(A,N-1); else d=xdet(A,ligne,N);

        // Affichage du resultat et basta :
        cout << "-----------------------------------------";
        cout<<"\nDeterminant : "<< d;
        cout << "\n-----------------------------------------\n\n";
        cout << "ASTUCE : Comparez les resultats obtenus a l'aide des
deux methodes !\n\n";
}
//////////////////////////////////////////////////////
```