

Correction TD 5 STL

L3, C++, FLIN 606

TdSTL

```
// ELEMENTS DE CORRECTION DU TP D'INTRODUCTION A STL
//-----

#include <iostream>
#include <list>
#include <algorithm>
#include <math.h>
using namespace std;
#include<iterator>
#include<vector>
#include<set>

//ACCES AUX TYPES INCLUS DANS LES CONTENEURS

template<typename C>
// le parametre doit etre un conteneur
// C::value_type doit etre un type muni de l'addition
// et 0 son  l ment neutre
typename C::value_type somme(const C&c)
{
    typename C::value_type s=0;
    typename C::const_iterator p = c.begin();
    while (p != c.end()){s+=*p; p++;}
    return s;
}

// CRIBLE D'ERATOSTHENE
// remarque : crible en francais, cribble en anglais

class divisiblePar { // classe d'objets fonction testant la divisibilit 
private:
    int diviseur;      // donn e membre : le diviseur
public:
    divisiblePar(int div) : diviseur(div) {}; // constructeur
    bool operator()(const int& dividende) {
        // surcharge de l'op rateur ()
        // les instances se comportent donc comme des fonctions
        return (dividende%diviseur)==0 ;
    }
};

void TesteDivisiblePar()
{
    divisiblePar f(2);
    divisiblePar g(3);

    cout << "f(4) = " << f(4) << endl;
    cout << "f(8) = " << f(8) << endl;
    cout << "f(5) = " << f(5) << endl;
}

template <class C, class Pred>
    // C est un conteneur, Pred est une fonction bool enne
    // prenant comme parametre unique un element de type C::value_type
```

```

    // ou un objet sur lequel on peut appliquer l'opérateur ()
void MMremove_if(C &c, Pred p) {
    typename C::iterator pr=remove_if(c.begin(),c.end(),p);
    // pr pointeur sur resultat
    // le remove_if ne supprime pas de cellules allouées du conteneur !
    // il retourne un itérateur sur la "fin" valide du conteneur
    // et ne change pas la valeur de c.end()
    // donc les "bons" elements de C se trouvent entre c.begin() et pr,
    // bizarre comme sémantique, non ?
    // il ne reste plus qu'à supprimer le reste du conteneur
    c.erase(pr,c.end()); // et voilà
}

template <class C>
void afficher(const C &c) { // afficher les elements d'un conteneur
    copy(c.begin(),c.end(),ostream_iterator<typename C::value_type>(cout," "));
    cout<< endl;
}

main() {
    // test de somme
    vector<int> v;
    v.push_back(1);v.push_back(2);v.push_back(3);v.push_back(4);v.push_back(5);
    cout << "somme des elements du vecteur =" << somme(v) << endl;
    set<float> s; s.insert(1);s.insert(2);s.insert(3);s.insert(4);s.insert(5);
    cout << "somme des elements du set =" << somme(s);

    // test du crible : on peut aussi faire une fonction
    list<int> crible; // crible vide
    int premier; // nombre premier
    int borne; // borne du crible
    cout<<endl<<"Entrez la borne du crible S.V.P. : "<<endl;
    cin>>borne;

    for (int i=2;i<=borne;i++) // remplissage
        crible.push_back(i); // crible = {2,3,4,...,borne}
    afficher(crible);

    while (!crible.empty()) { // tq pas vide
        cout<<(premier=crible.front())<<"/"; // afficher un nombre premier (le
        mini)
        crible.pop_front(); // extraire ce nombre premier (le
        mini)
        if (premier<=sqrt(borne)) // si ce n'est pas fini
            MMremove_if(crible,divisiblePar(premier));
            // supprimer tous les multiples de ce nombre premier
        else { // sinon
            afficher(crible); // afficher le reste qui ne contient que des
            premiers
            crible.erase(crible.begin(),crible.end());
        }
    }
}

```

CompteMots.cc

```
#include<iostream>
using namespace std;
#include<string>
#include<map>
//#include <pair.h>
#include <iterator>

ostream& operator<<(ostream& os, const pair<const string,int>& p)
{os<<p.first<<" "<<p.second<<endl; return os;}

int main()
{
    map<string, int, less<string> > dico;
    string Mot;

    while(cin>>Mot)
    {
        if (dico.find(Mot)!=dico.end())
            ++(*(dico.find(Mot))).second;
        else
            dico[Mot]=1;
    }

    map<string, int, less<string> >::iterator is;
    for(is=dico.begin();is!=dico.end();is++)
        cout<<*is<<" ";
    cout<<"\n_____ "<<endl;
}
```

testRemovelf

```
#include <iostream>
#include <list>
#include <algorithm>
#include <math.h>
using namespace std;
#include<iterator>
#include<vector>
#include<set>

bool odd (int a)
{ return a % 2;}

int numbers[6] = { 0, 0, 1, 1, 2, 2 };

int main ()
{
    int* ite =
    remove_if (numbers, numbers + 6, odd);
    for (int i = 0; i < 6; i++)
        cout << "adresse element " << i << " = " << (int)(&numbers[i])
            << " valeur = " << numbers[i] << endl;
    cout << endl;
    cout << "adresse retournée par remove_if =" << (int)ite;
    cout << endl;

    vector<int> v;
    v.push_back(1);v.push_back(2);v.push_back(3);v.push_back(4);v.push_back(5);
    v.push_back(6);
    vector<int>::iterator itev = remove_if (v.begin(), v.end() + 6, odd);
    for (int i = 0; i < 6; i++)
        cout << "adresse element " << i << " = " << (int)(&v[i])
            << " valeur = " << v[i] << endl;
    cout << endl;
    cout << "la dernière valeur désignée par remove_if est juste avant =" <<
*itev;
    cout << endl;

    return 0;
}
```