

Exemples du cours de programmation

Voilà je viens de terminer un recueil de tous les exemples réalisés par Mme Buseyne pendant le cours de programmation...

Normalement ils y sont tous... Si jamais vous en avez qui ne sont pas dessus prévenez-moi et je ferai les modifications.

Bonne nuit

Patasel

Déclaration d'une fonction :

```
#include <stdio.h>
int somme(int a,int b=2,int c=3,int d=4);
int main(){
    int res;
    int x=0,y=1,z=2,w=3;
    res=somme(x,y,z,w);
    printf("res=%d\n",res);
    res=somme(x,y,z);
    printf("res=%d\n",res);
    res=somme(x,y);
    printf("res=%d\n",res);
    res=somme(x);
    printf("res=%d\n",res);
return 0;
}
int somme(int a,int b,int c,int d){
    return a+b+c+d;
}
```

Les flots

```
#include <iostream.h>
int main(){
    int chiffre;
    char lettre;
    cout<<"Entrez un chiffre et une lettre"<<endl;
    cin>>chiffre>>lettre;
    cout<<"Le chiffre est : "<<chiffre<<endl;
    cout<<"La lettre est : "<<lettre<<endl;
    return 0;
}
```

Surcharge des fonctions

```
#include <stdio.h>
//int somme(int a,int b);
double somme(double a,double d);
int somme(int a=0,int b=0,int c=0);
int main(){
    int res;
```

```

        double dres;
        res=somme(1,5);
        printf("Res=%d\n",res);
        dres=somme(1.0,5.0);
        printf("dRes=%ld\n",dres);
//        res=somme(5,4.0);
//        printf("Res=%d\n",res);
        return 0;
    }
    /*int somme(int a,int b){
        return a+b;
    }*/
    double somme(double a,double b){
        return a+b;
    }
    int somme(int a,int b,int c){
        return a+b+c;
    }
}

```

Les classes :

```

#include <iostream.h>
#include <string.h>
class voiture{
    int vitesse;
    char marque[10];
public:
    void init(int vi,char ma[10])
    {
        vitesse=vi;
        strcpy(marque,ma);
    };
    void affiche()
    {
        cout<<"Vitesse="<<vitesse<<endl;
        cout<<"Marque="<<marque<<endl;
    };
};
void main(){
    voiture v1;
    v1.init(50,"VW");
    v1.affiche();
}

```

Fonction membre – Fonction globale :

```

#include <iostream.h>
int fonction(){
    return 0;
}
int j=1;

```

```

class ex3{
    int i;
public:
    int j;
    void initj(int jj){
        j=jj;
    }
    void init(int ii){
        i=ii;
    }
    int fonction(){
        return i;
    }
};
void main(){
    ex3 e;
    e.init(5);
    cout<<"Fonction membre="<<e.fonction()<<endl;
    cout<<"Fonction globale="<<::fonction()<<endl;
    e.initj(2);
    cout<<"j membre="<<e.j<<endl;
    cout<<"j globale="<<::j<<endl;
}

```

Constructeur – Destructeur :

1^{er} exemple :

```

#include <iostream.h>
#include <string.h>
class voiture{
    int vitesse;
    char marque[10];
public:
    voiture();
    voiture(int vi,char ma[10]);
    voiture(voiture &v);
    void affiche();
    voiture f(voiture v);
    ~voiture();
};
*voiture::voiture(){
    cout<<"Constructeur par défaut\n";
    vitesse=0;
    strcpy(marque,"VW");
}
voiture::voiture(int vi,char ma[10]){
    cout<<"Constructeur initialisation données membres\n";
    vitesse=vi;
    strcpy(marque,ma);
}

```

```

voiture::voiture(voiture& v){
    cout<<"constructeur par copie\n";
    vitesse=v.vitesse;
    strcpy(marque,v.marque);
}
voiture::~voiture(){
    cout<<"Appel du destructeur\n";
}
voiture voiture::f(voiture v){
    voiture tmp=v;
    return tmp;
}
void voiture::affiche(){
    cout<<"Vitesse="<<vitesse<<endl;
    cout<<"MARque="<<marque<<endl;
}
void main(){
    voiture v1;
    v1.affiche();
    voiture v2(70,"Mercedes");
    v2.affiche();
    voiture v3=v2;
    v3.affiche();
    v3.f(v2).affiche();
}

```

2^{ème} exemple :

```

#include <iostream.h>
#include <string.h>
class voiture{
    int vitesse;
    char marque[10];
public:
    //    voiture();
    voiture(int vi=0,char ma[10]="VW");
    void affiche();
    ~voiture();
};
/*voiture::voiture(){
    cout<<"Constructeur par défaut\n";
    vitesse=0;
    strcpy(marque,"VW");
}*/
voiture::voiture(int vi,char ma[10]){
    cout<<"Constructeur initialisation données membres\n";
    vitesse=vi;
    strcpy(marque,ma);
}
voiture::~voiture(){
    cout<<"Appel du destructeur\n";
}

```

```

}
void voiture::affiche(){
    cout<<"Vitesse="<<vitesse<<endl;
    cout<<"MARque="<<marque<<endl;
}
void main(){
    voiture v1;
    v1.affiche();
    voiture v2(70,"Mercedes");
    v2.affiche();
}

```

L'encapsulation :

```

#include <iostream.h>
#include <string.h>
//struct voiture{
class voiture{
    int vitesse;
public:
    void init(int vi,char ma[10]){
        vitesse=vi;
        strcpy(marque,ma);
    }
    void affiche(){
        cout<<"Vitesse="<<vitesse<<endl;
        cout<<"Marque="<<marque<<endl;
    }
private:
    char marque[10];
};
void main(){
    voiture v1;
    v1.init(50,"VW");
    v1.affiche();
}

```

Le mot clé STATIC :

```

#include <iostream.h>
class ex1{
    int i;
public:
    static int j;
    ex1(int ii);
    void affiche();
    static int getj();
};
int ex1::j=0;
ex1::ex1(int ii){
    i=ii;
}

```

```

}
void ex1::affiche(){
    cout<<"i="<<i<<endl;
}
int ex1::getj(){
    //return i;
    return j;
}
void main(){
    cout<<"j="<<ex1::j<<endl;
    ex1 e1(5);
    e1.affiche();
    cout<<ex1::getj()<<endl;

}

```

Surcharge des opérateurs :

1^{er} exemple :

```

#include <iostream.h>
class vecteur {
    float x;
    float y;
    float z;
public :
    vecteur(float x1=0,float y1=0,float z1=0);
    friend vecteur operator+(vecteur v1, vecteur v2);
    vecteur& operator=(vecteur& v);
    vecteur& operator +=(vecteur& v);
    vecteur operator --(int);
    vecteur& operator --();
    friend istream& operator>>(istream& tmp,vecteur& v);
    friend ostream& operator<<(ostream& tmp,vecteur v);
    void affiche();
};

```

```

vecteur::vecteur (float x1, float y1, float z1)
{
    x=x1;
    y=y1;
    z=z1;
}

```

```

vecteur operator+(vecteur v1, vecteur v2)
{
    vecteur tmp;
    tmp.x=v1.x+v2.x;
    tmp.y=v1.y+v2.y;
    tmp.z=v1.z+v2.z;
    return tmp;
}

```

```
}
```

```
vecteur& vecteur::operator=(vecteur& v)
```

```
{
```

```
    x=v.x;
```

```
    y=v.y;
```

```
    z=v.z;
```

```
    return *this;
```

```
}
```

```
vecteur& vecteur::operator +=(vecteur& v)
```

```
{
```

```
    x+=v.x;
```

```
    y+=v.y;
```

```
    z+=v.z;
```

```
    return *this;
```

```
}
```

```
vecteur vecteur::operator --(int){
```

```
    vecteur tmp=*this;
```

```
    tmp.x--;
```

```
    tmp.y--;
```

```
    tmp.z--;
```

```
    return tmp;
```

```
}
```

```
vecteur& vecteur::operator --(){
```

```
    x--;
```

```
    y--;
```

```
    z--;
```

```
    return *this;
```

```
}
```

```
istream& operator>>(istream& tmp, vecteur& v){
```

```
    cout<<"Entrez les valeurs";
```

```
    tmp>>v.x;
```

```
    tmp>>v.y;
```

```
    tmp>>v.z;
```

```
    return tmp;
```

```
}
```

```
ostream& operator<<(ostream& tmp,vecteur v) {
```

```
    tmp<<v.x;
```

```
    tmp<<v.y;
```

```
    tmp<<v.z;
```

```
    return tmp;
```

```
}
```

```
void vecteur::affiche()
```

```

{

    cout<<endl<<"x : "<<x<<endl;
    cout<<"y : "<<y<<endl;
    cout<<"z : "<<z<<endl;

}

```

```

void main()
{
    vecteur a(3,4,6);
    vecteur b(4,8,3);
    vecteur res;

    res=a+b;
    //res=operator +(a,b);

    res.affiche();

    a=b;
    a.affiche();

    a+=b;
    a.affiche();

    a--;
    a.affiche();

    --a;
    a.affiche();

    cin>>a;
    cout<<a;

```

```

}
2ème exemple :
#include "tableau.h"
tableau::tableau(int ta,int *t){
    taille=ta;
    tab=new int[taille];
    for(int i=0;i<taille;i++)
        tab[i]=t[i];
}
tableau::tableau(tableau& t){
    taille=t.taille;
    tab=new int[taille];
    for(int i=0;i<taille;i++)
        tab[i]=t.tab[i];

```

```

        cout<<"constructeur de copie\n";
    }
    void tableau::affiche(){
        for(int i=0;i<taille;i++)
            cout<<"tab= "<<tab[i]<<" ";
        cout<<endl;
    }
    //int tableau::operator[](int indice){
    int& tableau::operator[](int indice){
        return tab[indice];
    }
    //int tableau::operator()(int indice){
    int& tableau::operator()(int indice){
        return tab[indice];
    }
    void* tableau::operator new (size_t taille){
        cout<<"surcharge de new\n";
        return malloc(taille);
    }
    void tableau::operator delete(void *p){
        free(p);
        cout<<"surcharge de delete\n";
        return;
    }
    tableau::~tableau(){
        delete[] tab;
    }

```

Héritage simple :

1^{er} exemple :

```

#include <iostream.h>
#include <string.h>
class personne{
protected:
    char nom[20];
    char prenom[20];
    int age;
public:
    personne(char n[20]="",char p[20]="",int a=20);
    void affiche();
    ~personne();
};
class etudiant:public personne{
    int anneetude;
public:
    etudiant(int ae=2,char n[20]="",char p[20]="",int a=20);
    void affiche();
    ~etudiant();
};

```

```

personne::personne(char n[20],char p[20],int a){
    strcpy(nom,n);
    strcpy(prenom,p);
    age=a;
    cout<<"constructeur personne\n";
}
void personne::affiche(){
    cout<<"Nom:"<<nom<<endl;
    cout<<"Prénom:"<<prenom<<endl;
    cout<<"Age:"<<age<<endl;
}
personne::~~personne(){
    cout<<"destructeur de personne\n";
}
etudiant::etudiant(int ae,char n[20],char p[20],int a):personne(n,p,a){
    anneetude=ae;
    cout<<"constructeur etudiant\n";
}
void etudiant::affiche(){
    cout<<"Année etude:"<<anneetude<<endl;
    //personne::affiche();
    cout<<"Nom:"<<nom<<endl;
    cout<<"Prénom:"<<prenom<<endl;
    cout<<"Age:"<<age<<endl;
}
etudiant::~~etudiant(){
    cout<<"destructeur de etudiant\n";
}
void main(){
    personne p("Durant","Marcel",20);
    p.affiche();
    etudiant e(3,"Dupont","Charles",21);
    e.affiche();
    p=e;
    p.affiche();
    //e=p;
    personne *pp;
    etudiant *pe;
    pe=new etudiant(3,"Tilleul","Francois",20);
    pp=pe;
    pp->affiche();
    //pe=pp;
    pe=(etudiant*)pp;
    pe->affiche();
    pp=&p;
    pp->affiche();
    pp=&e;
    pp->affiche();
}

```

2^{ème} exemple :

```
#include <iostream.h>
#include <string.h>
class personne{
protected:
    char nom[20];
    char prenom[20];
    int age;
public:
    personne(char n[20],char p[20],int a);
    personne(personne& p);
    personne();
    void affiche();
    ~personne();
};
class etudiant:public personne{
    int anneetude;
public:
    etudiant(int ae,char n[20],char p[20],int a);
    void affiche();
    etudiant(etudiant& e);
    ~etudiant();
};
personne::personne(char n[20],char p[20],int a){
    strcpy(nom,n);
    strcpy(prenom,p);
    age=a;
    cout<<"constructeur personne\n";
}
personne::personne(){
    strcpy(nom,"");
    strcpy(prenom,"");
    age=20;
    cout<<"constructeur defaut personne\n";
}
personne::personne(personne& p){
    strcpy(nom,p.nom);
    strcpy(prenom,p.prenom);
    age=p.age;
    cout<<"constructeur copie personne\n";
}
void personne::affiche(){

    cout<<"Nom:"<<nom<<endl;
    cout<<"Prénom:"<<prenom<<endl;
    cout<<"Age:"<<age<<endl;
}
personne::~~personne(){
    cout<<"destructeur de personne\n";
```

```

}
etudiant::etudiant(int ae,char n[20],char p[20],int a):personne(n,p,a){
    anneetude=ae;
    cout<<"constructeur etudiant\n";
}
etudiant::etudiant(etudiant& e):personne(e){
    anneetude=e.anneetude;
    cout<<"constructeur copie etudiant\n";
}
void etudiant::affiche(){
    cout<<"Année etude:"<<anneetude<<endl;
    //personne::affiche();
    cout<<"Nom:"<<nom<<endl;
    cout<<"Prénom:"<<prenom<<endl;
    cout<<"Age:"<<age<<endl;
}
etudiant::~etudiant(){
    cout<<"destructeur de etudiant\n";
}
void main(){
    etudiant e(3,"Dupont","Charles",21);
    e.affiche();
    etudiant e1=e;
    e1.affiche();
}

```

3^{ème} exemple :

```

#include <iostream.h>
#include <string.h>
class personne{
protected:
    char nom[20];
    char prenom[20];
    int age;
public:
    personne(char n[20]="",char p[20]="",int a=3);
    personne(personne& p);
    personne& operator=(personne& p);
    //personne();
    void affiche();
    ~personne();
};
class etudiant:public personne{
    int anneetude;
public:
    etudiant(int ae=3,char n[20]="",char p[20]="",int a=20);
    void affiche();
    etudiant(etudiant& e);
    etudiant& operator=(etudiant& e);
    ~etudiant();
}

```

```

};
personne::personne(char n[20],char p[20],int a){
    strcpy(nom,n);
    strcpy(prenom,p);
    age=a;
    cout<<"constructeur personne\n";
}
personne& personne::operator =(personne& p){
    if(&p!=this){
        strcpy(nom,p.nom);
        strcpy(prenom,p.prenom);
        age=p.age;
    }

    cout<<"operator = personne"<<endl;
    return *this;
}
/*personne::personne(){
    strcpy(nom,"");
    strcpy(prenom,"");
    age=20;
    cout<<"constructeur defaut personne\n";
}*/
personne::personne(personne& p){
    strcpy(nom,p.nom);
    strcpy(prenom,p.prenom);
    age=p.age;
    cout<<"constructeur copie personne\n";
}
void personne::affiche(){

    cout<<"Nom:"<<nom<<endl;
    cout<<"Prénom:"<<prenom<<endl;
    cout<<"Age:"<<age<<endl;
}
personne::~~personne(){
    cout<<"destructeur de personne\n";
}
etudiant::etudiant(int ae,char n[20],char p[20],int a):personne(n,p,a){
    anneetude=ae;
    cout<<"constructeur etudiant\n";
}
etudiant::etudiant(etudiant& e):personne(e){
    anneetude=e.anneetude;
    cout<<"constructeur copie etudiant\n";
}
etudiant& etudiant::operator=(etudiant& e){
    if(&e!=this){
        anneetude=e.anneetude;
        strcpy(nom,e.nom);
    }
}

```

```

        strcpy(prenom,e.prenom);
        age=e.age;
    }
    return *this;
}
void etudiant::affiche(){
    cout<<"Année etude:"<<anneetude<<endl;
    //personne::affiche();
    cout<<"Nom:"<<nom<<endl;
    cout<<"Prénom:"<<prenom<<endl;
    cout<<"Age:"<<age<<endl;
}
etudiant::~~etudiant(){
    cout<<"destructeur de etudiant\n";
}
void main(){
    etudiant e(3,"Dupont","Charles",21);
    e.affiche();
    etudiant e1;
    e1=e;
    e1.affiche();
}

```

4^{ème} exemple :

```

#include <iostream.h>
#include <string.h>
class personne{
protected:
    char nom[20];
    char prenom[20];
    int age;
public:
    personne(char n[20]="",char p[20]="",int a=3);
    virtual void affiche();
    ~personne();
};
class etudiant:public personne{
    int anneetude;
public:
    etudiant(int ae=3,char n[20]="",char p[20]="",int a=20);
    void affiche();
    ~etudiant();
};
personne::personne(char n[20],char p[20],int a){
    strcpy(nom,n);
    strcpy(prenom,p);
    age=a;
    cout<<"constructeur personne\n";
}
void personne::affiche(){

```

```

        cout<<"Nom:"<<nom<<endl;
        cout<<"Prénom:"<<prenom<<endl;
        cout<<"Age:"<<age<<endl;
    }
    personne::~~personne(){
        cout<<"destructeur de personne\n";
    }
    etudiant::etudiant(int ae,char n[20],char p[20],int a):personne(n,p,a){
        anneeetude=ae;
        cout<<"constructeur etudiant\n";
    }
    void etudiant::affiche(){
        cout<<"Année etude:"<<anneeetude<<endl;
        //personne::affiche();
    //    cout<<"Nom:"<<nom<<endl;
        //cout<<"Prénom:"<<prenom<<endl;
        //cout<<"Age:"<<age<<endl;
    }
    etudiant::~~etudiant(){
        cout<<"destructeur de etudiant\n";
    }
    void main(){
        etudiant e(3,"Dupont","Charles",21);
        personne *p;
        personne p1("Durant","Marcel",20);
        //p=&p1;
        p=&e;
        p->affiche();
    }

```

Les fonctions virtuelles :

1^{er} exemple :

```

#include <iostream.h>
#include <string.h>
class vertebre{
public :
    virtual void manger()=0;
};
class poisson:public vertebre{
    char nom[20];
public:
    poisson(char n[20]="");
    void manger();
};
poisson::poisson(char n[20]){
    strcpy(nom,n);
}
void poisson::manger(){

```

```

        cout<<"Nopurriture = planctons\n";
    }
void main(){
//     vertebre v;
    vertebre *pv;
    poisson p("Sole");
    p.manger();
    pv=&p;
    pv->manger();
}

```

2^{ème} *exemple* :

```

#include <iostream.h>
class a{
    int *pa;
public:
    a(){
        pa=new int [2];
        cout<<"constructeur de a\n";
    }
    virtual void affiche(){
        cout<<"classe a\n";
    }
    virtual ~a(){
        cout<<"destructeur de a\n";
        delete pa;
    }
};
class b:public a{
    int *pb;
public:
    b(){
        pb=new int[2];
        cout<<"constructeur de b\n";
    }
    void affiche(){
        cout<<"classe b\n";
    }
    ~b(){
        cout<<"destructeur de b\n";
        delete pb;
    }
};
void main(){
    a *a1=new b;
    a1->affiche();
    delete a1;
}

```

L'amitié :

1^{er} exemple :

```
#include <iostream.h>
class base1{
    int b1;
public:
    base1(int b=0);
    void affiche();
    friend base1 inverse(base1 b);
};
base1::base1(int b){
    b1=b;
}
void base1::affiche(){
    cout<<"b1="<<b1<<endl;
}
base1 inverse(base1 b){
    return -b.b1;
}
void main(){
    base1 ba(5),res;
    res=inverse(ba);
    res.affiche();
    inverse(ba);
    ba.affiche();

}
```

2^{ème} exemple :

```
#include <iostream.h>
class base2;
class base1{
    int b1;
public:
    base1(int b=0);
    void affiche();
    void modifier_element(base2 *b, int bb2);
};
class base2{
    int b2;
public:
    base2(int bb2);
    void affiche();
    friend void base1::modifier_element(base2 *b, int bb2);

};
base1::base1(int b){
    b1=b;
}
void base1::affiche(){
    cout<<"b1="<<b1<<endl;
```

```

}
void base1::modifier_element(base2 *b, int bb2){
    b->b2=bb2;
}
base2::base2(int bb2){
    b2=bb2;
}
void base2::affiche(){
    cout<<"b2="<<b2<<endl;
}
void main(){
    base1 ba(5),res;
    ba.affiche();
    base2 b2(7);
    b2.affiche();
    ba.modifier_element(&b2,1);
    b2.affiche();
}

```

3^{ème} *exemple :*

```

#include <iostream.h>
class base2;
class base1{
    int b1;
public:
    base1(int b=0);
    void affiche();
    void modifier_element(base2 *b, int bb2);
};
class base2{
    int b2;
public:
    base2(int bb2);
    void affiche();
    friend void base1::modifier_element(base2 *b, int bb2);
};
base1::base1(int b){
    b1=b;
}
void base1::affiche(){
    cout<<"b1="<<b1<<endl;
}
void base1::modifier_element(base2 *b, int bb2){
    b->b2=bb2;
}
base2::base2(int bb2){
    b2=bb2;
}

```

```

void base2::affiche(){
    cout<<"b2="<<b2<<endl;
}
void main(){
    base1 ba(5),res;
    ba.affiche();
    base2 b2(7);
    b2.affiche();
    ba.modifier_element(&b2,1);
    b2.affiche();
}

```

3^{ème} exemple :

```

#include <iostream.h>
class base2;
class base1{
    int b1;
public:
    base1(int b=0);
    void affiche(base2 b);
    void modifier_element(base2 *b, int bb2);
};
class base2{
    int b2;
public:
    base2(int bb2);
    void affiche();
    friend class base1;
};
base1::base1(int b){
    b1=b;
}
void base1::affiche(base2 b){
    cout<<"b1="<<b1<<endl;
    cout<<"b2="<<b.b2<<endl;
}
void base1::modifier_element(base2 *b, int bb2){
    b->b2=bb2;
}
base2::base2(int bb2){
    b2=bb2;
}
void base2::affiche(){
    cout<<"b2="<<b2<<endl;
}
void main(){
    base1 ba(5);
    base2 b2(7);
    b2.affiche();
}

```

```

        ba.modifier_element(&b2,1);
        ba.affiche(b2);
    }

```

Les fichiers :

//exemple 1: flux entree et sortie séparés

```

#include <fstream.h>
#include <string.h>
#include <stdlib.h>
#include <iomanip.h>
#define MAX 20
class personne{
    char nom[MAX];
    char prenom[MAX];
public:
    personne(char n[MAX]="",char p[MAX]="");
    void creer_fichier();
    void lire_fichier();
    void ajouter_personne();
};
personne::personne(char n[MAX],char p[MAX]){
    strcpy(nom,n);
    strcpy(prenom,p);
}
void personne::creer_fichier(){
    ofstream sortie("pers.txt",ios::out);
    /*ofstream sortie;
    sortie.open("pers.txt",ios::out);*/
    if(sortie.fail()){
        cerr<<"Fail _ Impossible de créer le fichier pers.txt\n";
        exit(8);
    }
    if(sortie.bad()){
        cerr<<"Bad _ Impossible de créer le fichier pers.txt\n";
        exit(8);
    }
    //sortie<<setw(MAX)<<nom<<setw(MAX)<<prenom<<endl;
    personne p=*this;
    sortie.write((char*) &p,sizeof(personne));

    sortie.close();
}
void personne::lire_fichier(){
    //ifstream entree("pers.txt",ios::in);
    ifstream entree;
    entree.open("pers.txt",ios::in);
    if(entree.fail()){
        cerr<<"Fail _ Impossible de lire le fichier pers.txt\n";
        exit(8);
    }
}

```

```

    }
    if(entree.bad()){
        cerr<<"Bad _ Impossible de lire le fichier pers.txt\n";
        exit(8);
    }
    //while(entree.eof()!=1)
    /*while(!entree.eof()){
        entree>>nom>>prenom;
        if(entree.eof()!=1){
            cout<<"Nom:"<<nom<<endl;
            cout<<"Prénom:"<<prenom<<endl;
        }
    }*/

    personne p;
    while(entree.read((char*)&p,sizeof(personne))){
        cout<<"Nom:"<<p.nom<<endl;
        cout<<"Prénom:"<<p.prenom<<endl;
    }

    entree.close();
}

void personne::ajouter_personne(){
    ofstream sortie;
    sortie.open("pers.txt",ios::out|ios::app);
    if(sortie.fail()){
        cerr<<"Fail _ Impossible d'ouvrir le fichier pers.txt\n";
        exit(8);
    }
    if(sortie.bad()){
        cerr<<"Bad _ Impossible d'ouvrile fichier pers.txt\n";
        exit(8);
    }
    cout<<"Entrez le nom\n";
    cin>>nom;
    cout<<"Entrez le prénom\n";
    cin>>prenom;
    //sortie<<setw(MAX)<<nom<<setw(MAX)<<prenom<<endl;

    personne p=*this;;
    sortie.write((char*)&p,sizeof(personne));

    sortie.close();
}

void main(){
    personne p("NOM","PRENOM");
    p.creer_fichier();
    p.ajouter_personne();
    p.ajouter_personne();
    p.lire_fichier();
}

```

```
}
```

```
//exemple 2: flux fstream - entree et sortie ensemble
```

```
#include <fstream.h>
```

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

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

```
#include <iomanip.h>
```

```
#define MAX 20
```

```
class personne{
```

```
    char nom[MAX];
```

```
    char prenom[MAX];
```

```
public:
```

```
    personne(char n[MAX]="",char p[MAX]="");
```

```
    void creer_fichier();
```

```
    void lire_fichier();
```

```
    void ajouter_personne();
```

```
};
```

```
personne::personne(char n[MAX],char p[MAX]){
```

```
    strcpy(nom,n);
```

```
    strcpy(prenom,p);
```

```
}
```

```
void personne::creer_fichier(){
```

```
    fstream ensor;
```

```
    ensor.open("pers2.txt",ios::out);
```

```
    if(ensor.fail()){
```

```
        cerr<<"Fail _ Impossible de créer le fichier pers.txt\n";
```

```
        exit(8);
```

```
    }
```

```
    if(ensor.bad()){
```

```
        cerr<<"Bad _ Impossible de créer le fichier pers.txt\n";
```

```
        exit(8);
```

```
    }
```

```
    //sortie<<setw(MAX)<<nom<<setw(MAX)<<prenom<<endl;
```

```
    personne p=*this;
```

```
    ensor.write((char*) &p,sizeof(personne));
```

```
    ensor.close();
```

```
}
```

```
void personne::lire_fichier(){
```

```
    fstream ensor;
```

```
    ensor.open("pers2.txt",ios::in);
```

```
    if(ensor.fail()){
```

```
        cerr<<"Fail _ Impossible de lire le fichier pers.txt\n";
```

```
        exit(8);
```

```
    }
```

```
    if(ensor.bad()){
```

```
        cerr<<"Bad _ Impossible de lire le fichier pers.txt\n";
```

```
        exit(8);
```

```
    }
```

```
    //while(entree.eof()!=1)
```

```

    /*while(!entree.eof()){
        entree>>nom>>prenom;
        if(entree.eof()!=1){
            cout<<"Nom:"<<nom<<endl;
            cout<<"Prénom:"<<prenom<<endl;
        }
    }*/
    personne p;
    while(ensor.read((char*)&p,sizeof(personne))){
        cout<<"Nom:"<<p.nom<<endl;
        cout<<"Prénom:"<<p.prenom<<endl;
    }
    ensor.close();
}
void personne::ajouter_personne(){
    fstream ensor;
    ensor.open("pers2.txt",ios::app);
    if(ensor.fail()){
        cerr<<"Fail _ Impossible d'ouvrir le fichier pers.txt\n";
        exit(8);
    }
    if(ensor.bad()){
        cerr<<"Bad _ Impossible d'ouvrile fichier pers.txt\n";
        exit(8);
    }
    cout<<"Entrez le nom\n";
    cin>>nom;
    cout<<"Entrez le prénom\n";
    cin>>prenom;
    //sortie<<setw(MAX)<<nom<<setw(MAX)<<prenom<<endl;
    personne p=*this;;
    ensor.write((char*)&p,sizeof(personne));
    ensor.close();
}
void main(){
    personne p("NOM","PRENOM");
    p.creer_fichier();
    p.ajouter_personne();
    p.ajouter_personne();
    p.lire_fichier();
}

```

```

//fonction de positionnement dans un fichier
//fichier d'entree_sortie
#include <fstream.h>
#include <iostream.h>
void main(){
    char c;
    //écriture des données

```

```

fstream f("out.txt",ios::out|ios::in);
for(c=65;c<91;c++) f<<c;
f<<endl;
for(c=97;c<123;c++) f<<c;
f<<endl;
//f.close();
cout<<"écriture faite\n";
streampos ancpos=f.tellp(); //mémorise la position
f.seekp(0,ios::end);
cout<<"Taille du fichier :"<<f.tellp()<<" octet(s)\n";
f.seekp(ancpos,ios::beg);
//lecture,modification puis réécriture des données dans le fichier
f.seekg(0,ios::beg); //se positionne à 0 octets par rapport au début du fichier
c=f.get();
cout<<c;
while(c!=EOF){
    c=f.get();
    cout<<c;
}
f.close();
}

```

Strstream :

```

//convertir une chaîne en un nombre
//classe istrstream
#include <iostream.h>
#include <strstream.h>
#include <string.h>
//const int size = 100;
#define MAX 100
char buf[100] ="double 3.21 tableau chaine entier 321";

void main()
{
    istrstream istr(buf, MAX);
    cout << istr.str()<<endl; //convertit string en char* car dans ostream on n'a pas de surcharge de
    //ostream& operator <<(string )

    char arr[4][20];
    double d;
    long i;

    //semblable au sscanf
    istr >> arr[0] >> d >> arr[1] >> arr[2] >> arr[3] >> i;

    cout << arr[0] << " = " << d << "\n"
        << arr[1] << " = " << arr[2] << "\n"
        << arr[3] << " = " << i << endl;
    //autre exemple semblable à sscanf
    char temp[MAX]="1+1 vaut 2";
}

```

```

char c;
int i1,i2,i3;
char ch1[MAX];
istream istr1(temp,MAX);
istr1>>i1>>c>>i2>>ch1>>i3;
cout<<"i1="<<i1<<endl;
cout<<"i2="<<i2<<endl;
cout<<"i3="<<i3<<endl;
cout<<"c="<<c<<endl;
cout<<"ch1="<<ch1<<endl; //conversion chaîne en nombre (semblable à atol,atof)
char buffer[MAX]="123456";
istream istr2(buffer,MAX);
istr2>>i;
cout<<"i="<<i<<endl;
cout<<i+5<<endl;
strcpy(buffer,"12.3654");
istream istr3(buffer,MAX);
istr3>>d;
cout<<"d="<<d<<endl;
cout<<d+10.5<<endl;
}

```

//convertir un nombre en une chaîne de caractères

//exemple ostream

```
#include <iostream.h>
```

```
#include <strstrea.h>
```

```
#define MAX 20
```

```
void IntToString(int i,char *buffer,int len);
```

```
template<class T> void NombreToString(T t,char *buffer,int len);
```

```
int main()
```

```
{
```

```
    ostream out;
```

```
    out << "Exemple " << 1234<<endl;
```

```
    out << "la taille du tableau est de "
```

```
        << out.pcount() << " caracteres \n";
```

```
    out << "Utilisation ostream" << endl;
```

```
    out << "la taille finale du tableau est "
```

```
        <<out.pcount() << " caracteres \n"<<ends;
```

```
    //affichage de out
```

```
    cout << out.str() << endl;
```

//conversion de nombre en chaîne de caractères avec ostream

```
char nombre[MAX];
```

```
int i=12345;
```

```
IntToString(i,nombre,MAX);
```

```
double d=12.345;
```

```
NombreToString <double> (d,nombre,MAX);
```

//conversion plusieurs éléments en chaîne de caractères (semblable à sprintf)

```

    int i1=1,i2=1,i3=2;
    char c='+';
    char ch1[MAX]=" vaut ";
    ostream ostr1(nombre,MAX);
    ostr1<<i1<<c<<i2<<ch1<<i3<<ends;
    cout<<"Buffer="<<nombre<<endl;
    return 0;
}
//flux ostream ne fonctionne qu'avec les entiers
void IntToString(int i,char *buffer,int len){
    ostream out2(buffer,MAX);
    out2<<i<<ends;
    cout<<"Nombre="<<buffer<<endl;
}
//utiliser un patron de fonction pour type double par exemple
template<class T>
void NombreToString(T t,char *buffer,int len){
    ostream out2(buffer,MAX);
    out2<<t<<ends;
    cout<<"Nombre="<<buffer<<endl;
}

```

Istream et Ostream :

```

//classe ostream et istream
#include <ostream.h>
#include <istream.h>
void main(){
    //classe ostream
    cout.write("Ceci est un premier message",27);
    cout<<endl;
    cout.write("Ceci est un premier message",7);
    cout.put('\n');
    cout<<"Ceci est un deuxième message\n";
    cout<<15;
    cout<<"\n";
    cout.flush();
    //classe istream
    int res;
    char* ch=new char[50];
    cout<<"Entrez du texte\n";
    cin.getline(ch,20,'\n');
    cout<<ch<<endl;
    //ch[20]='\0';
    cout.flush();
    cout<<"Entrez du texte\n";
    cin.get(ch,10,'\n');
    cout<<ch<<endl;
    res=cin.gcount();
    cout<<"nombre octets lus="<<res<<endl;
}

```

```

    /*cout<<"entrez un texte\n";
    cin.read(ch,10);*/
    cout.flush();
    res=cin.peek();
    cout<<"res="<<res<<endl;
    cout<<"res="<<res<<endl;
    cout.flush();
    cin.clear();
    cout<<"Entrez un caractère\n";
    res=cin.get();
    cout<<"res="<<res<<endl;
    delete ch;
}

```

Formatage des informations :

```

//formatage de l'information
#include <iostream.h>
void main(){
int i = 245;//123456;
double d = 75.8901;
int hex=0xff;
//cout.precision(2);
cout.setf(ios::scientific, ios::floatfield);
cout << d<<endl; // écrit 7.59e+01

//afficher i en base octale
cout.setf(ios::oct,ios::basefield);
cout<<i<<endl;
cout.unsetf(ios::oct);
cout.setf(ios::hex,ios::basefield);
cout.setf(ios::left,ios::adjustfield);
cout<<hex<<endl;
cout.unsetf(ios::hex);

cout.width(7);
cout.fill('$');
cout << i<<endl; // écrit 245$$$$

cout.width(9);
cout.fill('#');
cout.setf(ios::left|ios::hex,
    ios::adjustfield|ios::basefield);
cout << i<<endl; // écrit f5#####

cout.fill(' ');
cout.width(6);
cout.setf(ios::internal|ios::showpos|ios::dec,
    ios::adjustfield|ios::showpos|ios::basefield);

```

```
cout << i<<endl; // écrit + 245
}
```

Les manipulateurs :

```
//manipulateurs
#include <iostream.h>
#include <iomanip.h>
void main(){
int i = 245;
double d = 75.8901;
int h=0xff;
cout.setf(ios::scientific, ios::floatfield);
cout <<setprecision(2)<< d<<endl; // écrit 7.59e+01
```

```
//afficher i en base octale
cout<<oct<<i<<endl;
cout<<resetiosflags(ios::oct);
cout.setf(ios::left,ios::adjustfield);
```

```
//afficher h en hexa
cout<<hex<<h<<endl;
cout<<resetiosflags(ios::hex);
```

```
cout.setf(ios::right,ios::adjustfield);
cout<<setw(7)<<setfill('$')<<i<<endl;// écrit $$$245
```

```
cout<<setiosflags(ios::left);
cout<<setw(9)<<setfill('#')<<hex<< i<<endl; // écrit f5#####
```

```
cout<<setiosflags(ios::internal|ios::showpos);
cout <<setfill(' ')<<setw(6)<<dec<< i<<endl; // écrit + 245
}
```

Création de manipulateurs :

```
//exemple 4 : création de manipulateur
#include <iostream.h>
#include <iomanip.h>
#include <limits.h> //pour ULONG_MAX
//manipulateur tab
ostream &tab(ostream& os){
    os<<'\t';
    return os;
}
```

```
//manipulateur de bin
ostream& bin(ostream &os,long val){
    unsigned long masque=~(ULONG_MAX >>1);
    while (masque){
        os<<((val & masque) ? '1':'0');
```

```

        masque >>=1;
    }
    return os;
}
//applicateur de bin
OMANIP(long bin(long val){
    return OMANIP(long) (bin,val);
}

void main(){
    cout<<14<<tab<<75<<endl;
    cout<<"2006 en binaire ="<<bin(2006)<<endl;
}

```

Autres exemples :

héritage :

```
#include <iostream.h>
```

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

```
class personne{
```

```
    char nom[20];
```

```
    char prenom[20];
```

```
    int age;
```

```
public:
```

```
    personne(char n[20]="",char p[20]="",int a=20);
```

```
    void affiche();
```

```
    ~personne();
```

```
};
```

```
class etudiant:public personne{
```

```
    int anneetude;
```

```
public:
```

```
    etudiant(int ae=2,char n[20]="",char p[20]="",int a=20);
```

```
    void affiche();
```

```
    ~etudiant();};
```

```
    personne::personne(char n[20], char p[20], int a){
```

```
        strcpy(nom,n);
```

```
        strcpy(prenom,p);
```

```
        age=a;
```

```
        cout<<"constructeur personne\n";
```

```
}
```

```
void personne::affiche(){
```

```
    cout<<"Nom: "<<nom<<endl;
```

```

        cout<<"Prenom: "<<prenom<<endl;
        cout<<"Age: "<<age<<endl;
    }

    personne::~~personne(){
        cout<<"Destructeur de personne\n";
    }

    etudiant::etudiant(int ae, char n[20], char p[20],int a):personne(n,p,a){
        anneeetude=ae;
        cout<<"Constructeur etudiant\n";
    }

    void etudiant::affiche(){
        cout<<"Annee etude: "<<anneeetude<<endl;
        personne::affiche();
    }

    etudiant::~~etudiant(){
        cout<<"Destructeur d'etudiant\n";
    }

    void main (){
        personne p("Fallon","Guillaume",25);
        p.affiche();
        etudiant e(2,"Touijrat","Abdel",29);
        e.affiche();
    }

```

Généricité :

```

//exemple string
#include <iostream>
#include <string>
using namespace std;
void main(){
    string s("123");
    s.resize(10);
    cout<<s<<endl;
    //taille
    cout<<"Nouvelle taille="<<s.length()<<endl;
    s.reserve(30);
    s="123456789";
    s[7]='4';
    s.at(6)='2';
    cout<<s<<endl;
    cout<<"élément 1:"<<s.data()[1]<<endl;

    string s2;
    char *p="4567890123";
    s2.assign(p+2,p+6);

```

```

cout<<s2<<endl;

s.assign(9,'Z');
cout<<s<<endl;

//concaténation
s+=s2;
cout<<s<<endl;

//utilisation de append
s.append("abcde12346",5);
cout<<s<<endl;

//extraction
string s3;
s3=s.substr(9,4);
cout<<"s3="<<s3<<endl;

//insertion de caractères
s.insert(3,"ABCD");
cout<<"s="<<s<<endl;

//suppression de caractères
s.erase(3,4);
cout<<"s="<<s<<endl;

//remplacement d'une sous-chaine dans une chaine
s.replace(0,5,"ABCDE");
cout<<"s="<<s<<endl;

//intervertir 2 chaînes
cout<<"s="<<s<<endl;
cout<<"s2="<<s2<<endl;
s.swap(s2);
cout<<"s="<<s<<endl;
cout<<"s2="<<s2<<endl;

//comparaison de chaines
if(s.compare(s2)>0) cout<<"s>s2\n";
else if (s.compare(s2)==0) cout<<"s=s2\n";
else cout<<"s<s2\n";
if(s>s2) cout<<"s>s2\n";
else if (s==s2) cout<<"s=s2\n";
else cout<<"s<s2\n";

//recherche dans une chaîne
string s4="abcde";
int pos=s2.find(s4);
cout<<"pos="<<pos<<endl;
pos=s2.rfind(s4);

```

```

        cout<<"pos="<<pos<<endl;

    }

//Ensemble
#include <iostream>
#include <set>
#include <algorithm>
using namespace std;
struct ltstr
{
    bool operator()(const char* s1, const char* s2) const
    {
        return strcmp(s1, s2) < 0;
    }
};

int main()
{
    const int N = 6;
    const char* a[N] = {"périphérique", "réseau", "programmation",
        "se", "électronique", "électricité"};
    const char* b[N] = {"logique", "micro", "électronique",
        "anglais", "programmation", "périphérique"};

    set<const char*, ltstr> A(a, a + N);
    set<const char*, ltstr> B(b, b + N);
    set<const char*, ltstr> C;

    cout << "Set A: ";
    copy(A.begin(), A.end(), ostream_iterator<const char*>(cout, " "));
    cout << endl;
    cout << "Set B: ";
    copy(B.begin(), B.end(), ostream_iterator<const char*>(cout, " "));
    cout << endl;

    cout << "Union: ";
    set_union(A.begin(), A.end(), B.begin(), B.end(),
        ostream_iterator<const char*>(cout, " "),
        ltstr());
    cout << endl;

    cout << "Intersection: ";
    set_intersection(A.begin(), A.end(), B.begin(), B.end(),
        ostream_iterator<const char*>(cout, " "),
        ltstr());
    cout << endl;
}

```

```

set_difference(A.begin(), A.end(), B.begin(), B.end(),
              inserter(C, C.begin()),
              Itstr());
cout << "Set C (difference of A and B): ";
copy(C.begin(), C.end(), ostream_iterator<const char*>(cout, " "));
cout << endl;
return 0;
}

```

```

//exception
#include <stdexcept>
#include <iostream>
#include <string>
using namespace std;
class rationnel{
    int num;
    int den;
public:
    rationnel(int n=0,int d=1);
    void setNum(int n);
    void setDen(int d) throw (invalid_argument);
    double estDouble() throw (range_error);
};
rationnel::rationnel(int n, int d) {
    num=n;
    den=d;
}
void rationnel::setNum(int n){
    num=n;
}
void rationnel::setDen(int d)
//throw (invalid_argument){
{    if (d==0) throw invalid_argument("Rationnel: Passage de dénominateur nul\n");
    den=d;
}

double rationnel::estDouble()
// throw (range_error){
{    if(den==0) throw range_error("Rationnel : division par 0\n");
    return num/den;
}
void main(){
    int denom;
    rationnel r1;
    try{
        rationnel r2(2,0);
        cout<<r2.estDouble();
    }catch(range_error &r){
        cerr<<"Exception range_error: "<<r.what()<<endl;
    }catch(std::exception &e){

```

```

        cerr<<"Exception: "<<e.what()<<endl;
    }

    do{
        cout<<"Entrez un denominateur "<<endl;
        cin>>denom;
        try{
            r1.setDen(denom);;
            cout<<r1.estDouble();
        }catch(invalid_argument &i ){
            cerr<<"Exception invalid_argument: "<<i.what()<<endl;
            continue;
        }catch(range_error &r){
            cerr<<"Exception range_error: "<<r.what()<<endl;
        }
        break;
    }while(1);
}

```

```

//Deque
#include <deque>
#include <iostream>
using namespace std;
typedef deque<int> int_deque;
void main(){
int_deque Q;//idem vector sauf insertion en plus en entête du vecteur
Q.push_back(3);
Q.push_front(1);
Q.insert(Q.begin() + 1, 2);
copy(Q.begin(), Q.end(), ostream_iterator<int>(cout, ";"));
cout<<endl;
//les valeurs 1 2 3 sont affichées
Q[2] = 0;
copy(Q.begin(), Q.end(), ostream_iterator<int>(cout, " "));
cout<<endl;
//les valeurs 1 2 0 sont affichées

//itération dans un deque
cout<<"affichage avec iterator\n";
int_deque::iterator it;
for(it=Q.begin();it!=Q.end();++it){
    int n=*it;
    cout<<n<<" ";
}
cout<<endl;
}

```

```

#include <iostream.h>
#include <queue> //file FIFO

```

```

#include <stack> //pile LIFO
#define MAX 10
using namespace std;
void main(){
    stack<int> st1; //pile - dernier entré premier sorti
    queue<int> qu1; //file -
    int j;
    for(j=0;j<MAX;j++){
        qu1.push(j); //ajoute j en fin de file
        st1.push(j); //ajoute j au sommet de la pile
    }
    cout<<"\nordre de sortie de la pile:\n";
    while(!st1.empty()){
        cout<<st1.top()<<" "; //élément au sommet pile
        st1.pop(); //retire élément au sommet pile
    }
    cout<<"\nordre de sortie de la file:\n";
    while(!qu1.empty()){
        cout<<qu1.front()<<" "; //élément en tête de file
        qu1.pop(); //retire élément en tête de file
    }
    cout<<endl;
}

```

```

//List

```

```

#include <iostream.h>
#include <list>
using namespace std;
typedef list <int> int_l;
void main(){
    int i;
    int_l ll1,ll2,ltemp;
    for(i=0;i<10;i++){
        ll1.push_front(i);
        ll1.push_back(i);
    }
    for(i=0;i<10;i++){
        ll2.push_front(i+1);
        ll2.push_back(i+1);
    }
    int_l::iterator it;
    cout<<"liste 1:\n";
    for(it=ll1.begin();it!=ll1.end();++it){
        int n=*it;
        cout<<n<<" ";
    }
    cout<<endl;
    cout<<"liste 2:\n";
    for(it=ll2.begin();it!=ll2.end();++it){
        int n=*it;

```

```

        cout<<n<<" ";
    }
    cout<<endl;
    ll2.sort();
    cout<<"liste 2 triée:\n";
    for(it=ll2.begin();it!=ll2.end();++it){
        int n=*it;
        cout<<n<<" ";
    }
    cout<<endl;
    ll2.swap(ll1);
    cout<<"liste 2 transformée en liste 1 et inversément:\n";
    for(it=ll2.begin();it!=ll2.end();++it){
        int n=*it;
        cout<<n<<" ";
    }
    cout<<endl;
    ltemp.swap(ll1);
    cout<<"liste ltemp transformée en liste 1:\n";
    cout<<"liste l1 transformée en liste ltemp (vide)\n";
    for(it=ltemp.begin();it!=ltemp.end();++it){
        int n=*it;
        cout<<n<<" ";
    }
    cout<<endl;

    cout<<"liste ll1\n";
    for(it=ll1.begin();it!=ll1.end();++it){
        int n=*it;
        cout<<n<<" ";
    }
    cout<<endl;

    ll2.splice(ll2.begin(),ltemp);
    cout<<"liste 2 modifiée:\n";
    for(it=ll2.begin();it!=ll2.end();++it){
        int n=*it;
        cout<<n<<" ";
    }
    cout<<endl;
    //echanger 2 liste
    ll1.swap(ltemp);
    //insertion ll2 dans ll1
    ll1.merge(ll2);
    cout<<"liste 1 est mergée avec 2\n";
    for(it=ll1.begin();it!=ll1.end();++it){
        int n=*it;
        cout<<n<<" ";
    }
    cout<<endl;

```

```

//inverse
ll1.reverse();
for(it=ll1.begin();it!=ll1.end();++it){
    int n=*it;
    cout<<n<<" ";
}
cout<<endl;

//compresse à une seule occurrence les valeurs identiques de la liste
ll1.unique();
for(it=ll1.begin();it!=ll1.end();++it){
    int n=*it;
    cout<<n<<" ";
}
cout<<endl;
}

```

```

//map et multimap
#include <iostream>
#include<map>
#include <string>
using namespace std;
void main(){
    //map
    typedef map<string,int> map_si;
    map_si map1;
    map_si::iterator it;
    string cle;
    int valeur;
    map1["janvier"]=31;
    map1["fevrier"]=28;
    map1["mars"]=31;
    map1["avril"]=30;
    map1["mai"]=31;
    map1["juin"]=30;
    map1["juillet"]=31;
    map1["aout"]=31;
    map1["septembre"]=30;
    map1["octobre"]=31;
    map1["novembre"]=30;
    map1["decembre"]=31;
    for(it=map1.begin();it!=map1.end();it++)
    {
        cle=(*it).first;
        valeur=(*it).second;
        cout<<cle<<" "<<valeur<<endl;
    }
    map<string,int>::iterator cur=map1.find("juin");
    map<string,int>::iterator prev=cur;
}

```

```

map<string,int>::iterator next=cur;
++next;
--prev;
cout<<"Précédent (en ordre alphabétique) est "<<(*prev).first<<endl;
cout<<"Suivant (en ordre alphabétique) est "<<(*next).first<<endl;

//multimap
typedef multimap<int,string> mmap_is;
mmap_is::iterator mit;
mmap_is mmap1;
mmap1.insert(mmap_is::value_type(2,"Deux"));
mmap1.insert(mmap_is::value_type(1,"Un"));
mmap1.insert(mmap_is::value_type(3,"Trois"));
mmap1.insert(mmap_is::value_type(1,"Een"));
mmap1.insert(mmap_is::value_type(2,"Deux"));
mmap1.insert(mmap_is::value_type(3,"Three"));
mmap1.insert(mmap_is::value_type(3,"Drei"));
mmap1.insert(mmap_is::value_type(4,"Quatre"));
//recherche un élément de clé 4 et l'affiche
mit=mmap1.find(4);
cout<<mit->first<<" : "<<mit->second<<endl;
//recherche le premier élément de clé 3
mit=mmap1.lower_bound(3);
//affiche tous les éléments de clé 3
while(mit != mmap1.upper_bound(3)){
    cout<<mit->first<<" : "<<mit->second<<endl;
    ++mit;
}
//recherche tous les éléments de clé 3
pair<mmap_is::iterator,mmap_is::iterator> p=mmap1.equal_range(3);
for(mit=p.first;mit!=p.second;++mit)
    cout<<mit->first<<" : "<<mit->second<<endl;
//efface élément de clé 2
mmap1.erase(2);
//affiche tous les éléments
for(mit=mmap1.begin();mit!=mmap1.end();mit++)
{
    cout<<mit->first<<" "<<mit->second<<endl;
}
}

//Patron de classe
#include <iostream.h>
template <class t>
class tableau{
    t tab[10];
public:
    tableau(t ta[10]);
    t operator[](int i);
    void affiche();
}

```

```

};
template <class t>
tableau<t>::tableau(t ta[10]){
    for(int i=0;i<10;i++)
        tab[i]=ta[i];
}
template <class t>
t tableau<t>::operator[](int i){
    return tab[i];
}
template <class t>
void tableau<t>::affiche(){
    for(int i=0;i<10;i++) cout<<tab[i]<<" ";
    cout<<endl;
}
class complexe{
    int reel,imag;
public:
    complexe (int r=0,int i=0);
    friend ostream& operator<<(ostream& o, complexe& c);
};
complexe::complexe(int r,int i){
    reel=r;
    imag=i;
}
ostream& operator<<(ostream& o, complexe& c){
    o<<c.reel<<" + i "<<c.imag<<endl;
    return o;
}
void main(){
    int t[10]={0,1,2,3,4,5,6,7,8,9};
    tableau <int> t1(t);
    t1.affiche();
    cout<<"element 3="<<t1[2]<<endl;
    char tt[10]='a','b','c','d','e','f','g','h','i','j';
    tableau <char> t2(tt);
    t2.affiche();
    int i;
    complexe ttt[10];
    for(i=0;i<10;i++) ttt[i]=complexe(i,i+1);
    tableau <complexe> t3(ttt);
    t3.affiche();
}

```

```
//Patron de classe 2
```

```
//Exemple 3 bis: méthodes templates dans une classe template
```

```
#include <iostream.h>
```

```
template <class t>
```

```
class tableau{
```

```
    t tab[10];
```

```

public:
    tableau(t ta[10]);
    t operator[](int i);
    template <class t2> int compare(t2 tt){
        int cpt;
        for(int i=0;i<10;i++){
            if (tab[i]==tt.tab[i]) cpt++;
        }
        if (cpt==10) return 1;
        else return 0;
    }
    void affiche();
};
template <class t>
tableau<t>::tableau(t ta[10]){
    for(int i=0;i<10;i++){
        tab[i]=ta[i];
    }
}
template <class t>
t tableau<t>::operator[](int i){
    return tab[i];
}
template <class t>
void tableau<t>::affiche(){
    for(int i=0;i<10;i++) cout<<tab[i]<<" ";
    cout<<endl;
}
void main(){
    int t[10]={0,1,2,3,4,5,6,7,8,9};
    int ttt[10]={10,11,12,13,14,15};
    tableau <int> t1(t);
    tableau <int> t3(ttt);
    t1.affiche();
    cout<<"element 3="<<t1[2]<<endl;

    if(t1.compare(t3)) cout<<"les tableaux sont identiques\n";
    else cout<<"Les tableaux sont différents\n";

    char tt[10]={'a','b','c','d','e','f','g','h','i','j'};
    tableau <char> t2(tt);
    t2.affiche();

}

```

```

//Patron de classe 3
//patron de classe dérivant d'un patron de classe
#include <iostream.h>
#include <string.h>
template <class t>

```

```

class tableau{
    t tab[10];
public:
    tableau(t ta[10]);
    t operator[](int i);
    void affiche();
};
template <class t>
tableau<t>::tableau(t ta[10]){
    for(int i=0;i<10;i++)
        tab[i]=ta[i];
}
template <class t>
t tableau<t>::operator[](int i){
    return tab[i];
}
template <class t>
void tableau<t>::affiche(){
    for(int i=0;i<10;i++) cout<<tab[i]<<" ";
    cout<<endl;
}
template <class u,class t>
class tableaunominatif:public tableau<t>{
    u nom;
public:
    tableaunominatif(u n,t tt[10]);
    void affiche();
};
template<class u,class t>
tableaunominatif<u,t>::tableaunominatif(u n,t tt[10]):tableau<t>(tt){
    strcpy(nom,n);
}
template<class u,class t>
void tableaunominatif<u,t>::affiche(){
    cout<<nom<<endl;
}
void main(){
    int t[10]={0,1,2,3,4,5,6,7,8,9};
    tableau <int> t1(t);
    t1.affiche();
    tableaunominatif <char[10],int> tn1("tn1",t);
    tn1.affiche();
    char tt[10]='a','b','c','d','e','f','g','h','i','j';
    tableau <char> t2(tt);
    t2.affiche();
    tableaunominatif <char[10],char> tn2("tn2",tt);
    tn2.affiche();
}

```

```

//Patron de classe 4
//patron de classe dérivant d'une classe ordinaire
#include <iostream.h>
#include <string.h>
class tableau{
    int tab[10];
public:
    tableau(int ta[10]);
    int operator[](int i);
    void affiche();
};
tableau::tableau(int ta[10]){
    for(int i=0;i<10;i++)
        tab[i]=ta[i];
}
int tableau::operator[](int i){
    return tab[i];
}
void tableau::affiche(){
    for(int i=0;i<10;i++) cout<<tab[i]<<" ";
    cout<<endl;
}
//patron dérivant de la classe tableau
template <class u>
class tableaunominatif:public tableau{
    u nom;
public:
    tableaunominatif(u n,int tt[10]);
    void affiche();
};
template<class u>
tableaunominatif<u>::tableaunominatif(u n,int tt[10]):tableau(tt){
    nom=n;
}
tableaunominatif<char[10]>::tableaunominatif(char n[10],int tt[10]):tableau(tt){
    strcpy(nom,n);
}
template<class u>
void tableaunominatif<u>::affiche(){
    cout<<nom<<endl;
}
void main(){
    int t[10]={0,1,2,3,4,5,6,7,8,9};
    tableau t1(t);
    t1.affiche();
    tableaunominatif <char[10]> tn1("tn1",t);
    tn1.affiche();
    tableaunominatif <char> tn2('2',t);
    tn2.affiche();
}

```

```

}
//Patron de classe
//classe ordinaire dérivant d'un patron de classe
#include <iostream.h>
#include <string.h>
template <class t>
class tableau{
    t tab[10];
    public:
        tableau(t tt[10]);
        t operator[](int i);
        void affiche();
};
template <class t>
tableau<t>::tableau(t tt[10]){
    for(int i=0;i<10;i++)
        tab[i]=tt[i];
}
template <class t>
void tableau<t>::affiche(){
    for(int i=0;i<10;i++) cout<<tab[i]<<" ";
    cout<<endl;
}
template <class t>
t tableau<t>::operator[](int i){
    return tab[i];
}
//classe ordinaire dérivant d'un patron de classe
class tableaunominatif:public tableau<char>{
    char nom[10];
    public:
        tableaunominatif(char n[10],char tt[10]);
        void affiche();
};
tableaunominatif::tableaunominatif(char n[10],char tt[10]):tableau<char>(tt){
    strcpy(nom,n);
}
void tableaunominatif::affiche(){
    tableau<char>::affiche();
    cout<<"nom="<<nom<<endl;
}
void main(){
    int tab1[10]={0,1,2,3,4,5,6,7,8,9};
    tableau <int> t1(tab1);
    t1.affiche();
    cout<<"element 3="<<t1[2]<<endl;
    //avec caractères
    //char tab2[10]='a','b','c','d','e','f','g','h','i','j';
    char tab2[10]="abcdefgh";
}

```

```

        tableau <char> t2(tab2);
        t2.affiche();
        cout<<"element 4="<<t2[3]<<endl;
        tableaunominatif tn2("tn2",tab2);
        tn2.affiche();
    }
}
//Patron de classe
#include <iostream.h>
#include <string.h>
class tableau{
    int tab[10];
    public:
        tableau(int tt[10]);
        int operator[](int i);
        void affiche();
};
tableau::tableau(int tt[10]){
    for(int i=0;i<10;i++)
        tab[i]=tt[i];
}
void tableau::affiche(){
    for(int i=0;i<10;i++) cout<<tab[i]<<" ";
    cout<<endl;
}
int tableau::operator[](int i){
    return tab[i];
}
//patron de classe dérivant d'une classe ordinaire
template <typename u>
class tableaunominatif:public tableau{
    u nom;
    public:
        tableaunominatif(u n,int tt[10]);
        // tableaunominatif(char *n,int tt[10]);
        void affiche();
};
template <typename u>
tableaunominatif<u>::tableaunominatif(u n,int tt[10]):tableau(tt){
    nom=n;
}
tableaunominatif<char [10]>::tableaunominatif(char n[10],int tt[10]):tableau(tt){
    strcpy(nom,n);
}
template <typename u>
void tableaunominatif<u>::affiche(){
    tableau::affiche();
    cout<<"nom="<<nom<<endl;
}
void main(){

```

```

    int tab1[10]={0,1,2,3,4,5,6,7,8,9};
    tableau t1(tab1);
    t1.affiche();
    cout<<"element 3="<<t1[2]<<endl;
    //derivation patron patron
    tableaunominatif <int> tn1(1,tab1);
    tn1.affiche();
    tableaunominatif <char> tn2('t',tab1);
    tn2.affiche();
    tableaunominatif <char[10]> tn3("test",tab1);
    tn3.affiche();

}

//Patron de classe
//classe ordinaire dérivant d'un patron de classe
#include <iostream.h>
#include <string.h>
template <class t>
class tableau{
    t tab[10];
    public:
        tableau(t tt[10]);
        t operator[](int i);
        void affiche();
};
template <class t>
tableau<t>::tableau(t tt[10]){
    for(int i=0;i<10;i++)
        tab[i]=tt[i];
}
template <class t>
void tableau<t>::affiche(){
    for(int i=0;i<10;i++) cout<<tab[i]<<" ";
    cout<<endl;
}
template <class t>
t tableau<t>::operator[](int i){
    return tab[i];
}
class tableaunominatif:public tableau<char>{
    char nom[10];
    public:
        tableaunominatif(char n[10],char tt[10]);
        void affiche();
};
tableaunominatif::tableaunominatif(char n[10],char tt[10]):tableau<char>(tt){
    strcpy(nom,n);
}
void tableaunominatif::affiche(){

```

```

        tableau<char>::affiche();
        cout<<"nom="<<nom<<endl;
    }
void main(){
    int tab1[10]={0,1,2,3,4,5,6,7,8,9};
    tableau <int> t1(tab1);
    t1.affiche();
    cout<<"element 3="<<t1[2]<<endl;
    //avec caractères
    //char tab2[10]={'a','b','c','d','e','f','g','h','i','j'};
    char tab2[10]="abcdefgh";
    tableau <char> t2(tab2);
    t2.affiche();
    cout<<"element 4="<<t2[3]<<endl;
    tableaunominatif tn2("tn2",tab2);
    tn2.affiche();
}

```

```

//Patron de fonction
#include <iostream.h>
template <class type>
type carre(type a){
    return a*a;
}
void main(){
    int n=5;
    cout<<"carré de "<<n<<"="<<carre(n)<<endl;
    double d=5.6;
    cout<<"carré de "<<d<<"="<<carre(d)<<endl;
}

```

```

//Patron de fonction
#include <iostream.h>
#include <string.h>
template <typename t>
int egal(t a,t b){
    return(a==b?1:0);
}
//spécialisation
int egal(char *s1,char *s2){
    return(strcmp(s1,s2)==0?1:0);
}
//surcharge
template <typename t>
int egal(t a,t b,t c){
    return (a==b && b==c ?1:0);
}

void main(){
    if(egal(10,20)) cout <<"les nombres sont égaux\n";
    else cout<<"Les nombres sont différents\n";
}

```

```

    if(egal('c','c')) cout<<"les caractères sont égaux\n";
    else cout<<"les caractères sont différents\n";
    if(egal("petit","grand")) cout<<"les chaines de caractères sont égaux\n";
    else cout<<"les chaines de caractères sont différents\n";
    if(egal(10,10,20)) cout <<"les nombres sont égaux\n";
    else cout<<"Les nombres sont différents\n";
    if(egal('c','c','c')) cout<<"les caractères sont égaux\n";
    else cout<<"les caractères sont différents\n";
}

```

```

//Priority
//priority_queue
#include <iostream>
#include <queue>
using namespace std;
void main(){
    typedef priority_queue <int> pq_i;
    pq_i q;
    q.push(5);
    q.push(10);
    q.push(15);
    while(!q.empty()){
        cout<<q.top()<<endl;
        q.pop();
    }
}

```

```

//Set
//set et multiset
#include <iostream>
#include <set>
#include <algorithm>
using namespace std;
void main(){
    typedef set<int> set_i;
    set_i::iterator it;
    set_i ens1,ens2;
    int i;
    for(i=0;i<10;i++)
        ens1.insert(i);
    for(i=5;i<15;i++)
        ens2.insert(i);
    for (it=ens1.begin();it!=ens1.end();it++)
        cout<<*it<<" ";
    cout<<endl;
    for (it=ens2.begin();it!=ens2.end();it++)
        cout<<*it<<" ";
    cout<<endl;
    //intersection
    cout<<"Intersection:\n";
    set_intersection(ens1.begin(),ens1.end(),ens2.begin(),ens2.end(),ostream_iterator<int>(cout," "));
}

```

```

    cout<<endl;
    //union
    cout<<"Union:\n";
    set_union(ens1.begin(),ens1.end(),ens2.begin(),ens2.end(),ostream_iterator<int>(cout," "));
    cout<<endl;
    //différence
    cout<<"Différence:\n";
    set_difference(ens1.begin(),ens1.end(),ens2.begin(),ens2.end(),ostream_iterator<int>(cout," "));
    cout<<endl;
    //symetric différence
    cout<<"Symetric Différence:\n";
    set_symmetric_difference(ens1.begin(),ens1.end(),ens2.begin(),ens2.end(),ostream_iterator<int>(cou
t," "));
    cout<<endl;

    //multiset
    typedef multiset<char> mset_c;
    mset_c::iterator mit;
    mset_c mens1;
    mens1.insert('k');
    mens1.insert('a');
    mens1.insert('k');
    mens1.insert('l');
    mens1.insert('c');
    for (mit=mens1.begin();mit!=mens1.end();mit++)
        cout<<*mit<<" ";
    cout<<endl;
}

```

```

//Vector
#include <iostream>
#include <vector>
#include <algorithm>
#include <string.h>
#include <stdlib.h>
#define MAX 10
using namespace std;
void main(){
    int i;
    //création d'un vecteur d'entiers de MAX éléments
    vector <int> vect(MAX);
    //remplir le vecteur
    for(i=0;i<MAX;i++)
        vect.at(i)=i;
    //élément 5 fait-il parti du vecteur ?
    cout<<"5 fait -il parti du vecteur ?\n";
    vector<int>::iterator result = find(vect.begin(), vect.end(), 5);
    cout<<(*result==5?"oui":"non")<<endl;

    //afficher le contenu du vecteur

```

```

for(i=0;i<vect.size();i++)
    cout<<vect.at(i)<<" ";
cout<<endl;
//ajout d'un élément à la fin du vecteur
vect.push_back(10);
cout<<"taille de vect="<<vect.size()<<endl;
//enlever le dernier élément
vect.pop_back();
cout<<"taille de vect="<<vect.size()<<endl;
cout<<"Capacité="<<vect.capacity()<<endl;

//création d'un vecteur de 2 chaînes
vector <char*> vectc(2);
vectc.at(0)="mot0";
vectc.at(1)="mot1";
cout<<"Taille de vectc="<<vectc.size()<<endl;
//afficher le vecteur
for(i=0;i<vectc.size();i++)
    cout<<vectc.at(i)<<" ";
cout<<endl;
//ajout d'un élément à la fin
vectc.push_back("mot2");
cout<<"Taille de vectc="<<vectc.size()<<endl;
//suppression de tous les éléments
vectc.clear();
cout<<"Taille de vectc="<<vectc.size()<<endl;

//échanger 2 vecteurs
vector <int> vect2(MAX);
//remplir le vecteur
for(i=0;i<MAX;i++)
    vect2.at(i)=i+5;
//afficher le vecteur
cout<<"vect2:\n";
for(i=0;i<vect2.size();i++)
    cout<<vect2.at(i)<<" ";
cout<<endl;
//échange de vect et vect2
vect.swap(vect2);
//afficher les vecteurs
cout<<"vect2:\n";
for(i=0;i<vect2.size();i++)
    cout<<vect2.at(i)<<" ";
cout<<endl;
cout<<"vect:\n";
for(i=0;i<vect.size();i++)
    cout<<vect.at(i)<<" ";
cout<<endl;

//afficher le dernier élément de vect

```

```

cout<<"dernier element de vect:"<<vect.back()<<endl;

//afficher le premier élément de vect
cout<<"premier element de vect:"<<vect.front()<<endl;

//itérateur dans un vecteur
typedef vector<int> v_int;
v_int vv;
for(i=0;i<MAX;i++)
    vv.push_back(i);
v_int::iterator it;
for(it=vv.begin();it!=vv.end();++it)
{
    int n=*it;
    cout<<"n="<<n<<" ";
}
cout<<endl;
//insertion d'un élément
it=vv.begin();
vv.insert(it,20);
//affichage du vecteur
for(it=vv.begin();it!=vv.end();++it)
{
    int n=*it;
    cout<<"n="<<n<<" ";
}
cout<<endl;
}

```