```
#include<iostream>
#include<string>
#include<map>
using namespace std;


class ProduitChocolaterie
{
private:
        string reference;
        float poids; // en kilo
public:
        ProduitChocolaterie();
        ProduitChocolaterie(string r, float p);
        virtual ~ProduitChocolaterie();
        virtual string getReference()const;
        virtual void setReference(string r);
        virtual float getPoids()const;
        virtual void setPoids(float p);
        virtual float prix()const=0;
        virtual void affiche(ostream&)const;
};


ostream& operator<<(ostream& flot, const ProduitChocolaterie& p)
{p.affiche(flot); return flot;}

ProduitChocolaterie::ProduitChocolaterie(){reference="Produit inconnu";
poids=0;cout <<"ProduitChocolaterie";}
ProduitChocolaterie::ProduitChocolaterie(string r, float p){reference=r;
poids=p;}
ProduitChocolaterie::~ProduitChocolaterie(){}
string ProduitChocolaterie::getReference()const{return reference;}
void ProduitChocolaterie::setReference(string r){reference=r;}
float ProduitChocolaterie::getPoids()const{return poids;}
void ProduitChocolaterie::setPoids(float p){poids=p;}
void ProduitChocolaterie::affiche(ostream& f)const
{f<<"ref="<<reference<<endl<<"poids="<<poids<<"
kg"<<endl<<"prix="<<prix()<<endl;}

class Moulage : public virtual  ProduitChocolaterie
//class Moulage : public   ProduitChocolaterie
{
private:
        float prixKilo;
        float volume; // en cm3
public:
        Moulage();
        Moulage(string r, float p, float pk, float v);
        virtual ~Moulage();
        virtual float getPrixKilo()const;
        virtual void setPrixKilo(float p);
        virtual float getVolume()const;
        virtual void setVolume(float v);
        virtual float prix()const;
        virtual void affiche(ostream&)const;
};

Moulage::Moulage(){prixKilo=0; volume=0; cout <<"Moulage";}
Moulage::Moulage(string r, float p, float pk, float
v):ProduitChocolaterie(r,p)
```

```
{prixKilo=pk; volume=v;}
Moulage::~Moulage(){}
float Moulage::getPrixKilo()const{return prixKilo;}
void Moulage::setPrixKilo(float p){prixKilo=p;}
float Moulage::getVolume()const{return volume;}
void Moulage::setVolume(float v){volume=v;}
// prix = prix chocolat + 3 euros pour chaque 10cm3 de volume
float Moulage::prix()const{return prixKilo*getPoids()+((int)volume/10)*3;}
void Moulage::affiche(ostream&f)const
{ProduitChocolaterie::affiche(f);f<<"prixkilo="<<prixKilo<<endl<<"volume="<
<volume<<endl;}

class Specialite  : public virtual ProduitChocolaterie
//class Specialite  : public  ProduitChocolaterie
{
private:
        float prixBoite;
public:
        Specialite();
        Specialite(string r, float p, float pr);
        virtual ~Specialite();
        virtual float prix()const;
        virtual void setPrix(float p);
};

Specialite::Specialite(){cout <<"Specialite";}
Specialite::Specialite(string r, float p, float
pr):ProduitChocolaterie(r,p){prixBoite=pr;}
Specialite::~Specialite(){}
float Specialite::prix()const{return prixBoite;}
void Specialite::setPrix(float p){prixBoite=p;}

// Question hŽritage multiple
// forme instance + ordre appel des constructeurs
// sur quelles mŽthodes y a-t-il conflit ?
// dans les cas rŽpŽtŽ et virtuels

class OursDeBerne : virtual public Specialite, virtual public Moulage
{
private :
        bool fourrageCreme;
public:
        OursDeBerne();
        OursDeBerne(string r, float p, float pr, float pk, float v, bool
f);
        virtual ~OursDeBerne();
        virtual float prix()const;
};

OursDeBerne::OursDeBerne(){cout <<"Oursdeberne";}
OursDeBerne::OursDeBerne(string r, float p, float pr, float pk, float v,
bool f)
:Specialite(r,p,pr),Moulage(r,p,pk,v),ProduitChocolaterie(r,p)
{fourrageCreme=f;}
OursDeBerne::~OursDeBerne(){}
float OursDeBerne::prix()const
{ //cout << Specialite::prix() << "   " << Moulage::prix() <<endl;
        if (Specialite::prix() > Moulage::prix())
        return Specialite::prix();
        else return Moulage::prix();
}
```

```
/* // version non parametree
class Panier
{
private:
        map<ProduitChocolaterie *,int> contenu;
public:
        Panier();
        virtual ~Panier();
        virtual void ajoute(ProduitChocolaterie *p, int qte);
        virtual void affiche(ostream& f);
        virtual float prix();
};

Panier::Panier(){}
Panier::~Panier(){}
void Panier::ajoute(ProduitChocolaterie *p, int qte)
{contenu[p]=qte;}
void Panier::affiche(ostream& f)
{
        for (map<ProduitChocolaterie *,int>::iterator it=contenu.begin();it
!= contenu.end(); it++)
                f << (*it).first->getReference() << "  " << (*it).second <<
endl;
}
float Panier::prix()
{
        float prixTotal=0;
        for (map<ProduitChocolaterie *,int>::iterator it=contenu.begin();it
!= contenu.end(); it++)
                {prixTotal+=((*it).first->prix()) * ((*it).second) ;}
        return prixTotal;
}
*/

// version parametree

template<typename TypeProduit>
class Panier
{
private:
        map<TypeProduit,int> contenu;
public:
        Panier();
        virtual ~Panier();
        virtual void ajoute(TypeProduit p, int qte);
        virtual void affiche(ostream& f);
        virtual float prix();
        virtual bool present(TypeProduit p);
};

class ErreurElementDejaPresent
{
};


template<typename TypeProduit>
Panier<TypeProduit>::Panier(){}
template<typename TypeProduit>
Panier<TypeProduit>::~Panier(){}
template<typename TypeProduit>
```

```
bool Panier<TypeProduit>::present(TypeProduit p)
{
        for (typename map<TypeProduit,int>::iterator it=contenu.begin();it
!= contenu.end(); it++)
                if ((*it).first==p)
                                return true;
}
template<typename TypeProduit>
void Panier<TypeProduit>::ajoute(TypeProduit p, int qte)
{
        if (contenu.find(p)==contenu.end())
                contenu[p]=qte;
        else {throw new ErreurElementDejaPresent();}
}

template<typename TypeProduit>
void Panier<TypeProduit>::affiche(ostream& f)
{
        for (typename map<TypeProduit,int>::iterator it=contenu.begin();it
!= contenu.end(); it++)
                f << (*it).first->getReference() << "  " << (*it).second <<
endl;
}
template<typename TypeProduit>
float Panier<TypeProduit>::prix()
{
        float prixTotal=0;
        for (typename map<TypeProduit,int>::iterator it=contenu.begin();it
!= contenu.end(); it++)
                {prixTotal+=((*it).first->prix()) * ((*it).second) ;}
        return prixTotal;
}


int main()
{
        ProduitChocolaterie *pc = new Moulage("Lapin",0.1,40,10);
        pc->affiche(cout); cout << endl; cout << *pc << endl;
        ProduitChocolaterie *g = new Specialite("Gavotte",0.1,15.0);
        g->affiche(cout); cout << endl; cout << *g << endl;
        cout <<"------ debut construction ours de berne" << endl;
        OursDeBerne *oberne = new OursDeBerne();
        cout <<"------ fin construction ours de berne" << endl;
        OursDeBerne *ob = new OursDeBerne("ours de berne",0.3,1,30,0.2,
true);

        //ob->Specialite::affiche(cout); //cout << *ob << endl;
        Panier<ProduitChocolaterie*> p;
        p.ajoute(pc,2); p.ajoute(g,1); p.ajoute(ob,1);
        p.affiche(cout); cout << endl; cout << "Prix total=" << p.prix() <<
endl;
        try{p.ajoute(pc,2);
        }
        catch(ErreurElementDejaPresent *e) {cout << "element deja
present"<< endl;}
}
```

```
// P1 differe d'un parcours df-glouton
// en cas d'arcs de transitivite
// heritage virtuel
// exemple de catastrophe avec une copie dans un historique
// avant la construction de l'objet

#include<iostream.h>

/*
 class
 {
 public:
 ();
 virtual ~();
 };

 ::(){cout << "  ";}
 ::~(){cout << "  ";}
*/


 typedef int Vine; // but should be a class
 #define Historic cout

 class Wine
 {

 protected:
 Vine *Composition;
 int NbVine;
 public:
 Wine();
 virtual ~Wine();
 virtual ostream& copyStream(ostream&)const;
 };

Wine::Wine(){cout << " WineConstruction" << endl; Composition=NULL;
NbVine=0;}
 Wine::~Wine(){cout << " ~Wine ";}

 ostream& Wine::copyStream(ostream& os)const
 {  int i;
    for (i=0; i<NbVine; i++)
      {os << "vine " << i << Composition[i] << endl;}
    return os;
 }


class Wine_Invention : virtual public Wine
 {
 public:
 Wine_Invention();
 virtual ~Wine_Invention();
 };

 Wine_Invention::Wine_Invention()
 {
  cout << " Wine_InventionConstruction" << endl;
```

```
  cout << " How many vine are mixed ? ";
  cin >> NbVine; Composition = new Vine[NbVine];
  int i;
  for (i=0; i<NbVine; i++)
    {cout << "vine " << i << " ? "; cin >> Composition[i];}
  }


 Wine_Invention::~Wine_Invention(){cout << " ~Wine_Invention " << endl;}


 class Supervised_Wine : virtual public Wine
 {
 public:
 Supervised_Wine();
 virtual ~Supervised_Wine();
 };

 Supervised_Wine::Supervised_Wine()
           {cout << " Supervised_WineConstruction " << endl;
                                  copyStream(Historic);}
 Supervised_Wine::~Supervised_Wine(){cout << " ~Supervised_Wine ";}

 class SupervisedBurgundy
   : virtual public Wine_Invention, virtual public Supervised_Wine
 {
 public:
 SupervisedBurgundy();
 virtual ~SupervisedBurgundy();
 };

 SupervisedBurgundy::SupervisedBurgundy()
{cout << " SupervisedBurgundyConstruction " << endl;}
 SupervisedBurgundy::~SupervisedBurgundy()
{cout << " ~SupervisedBurgundy ";}


 class Supervised_Cotes_De_Beaune
   : virtual public Supervised_Wine, virtual public SupervisedBurgundy
 {
 public:
 Supervised_Cotes_De_Beaune();
 virtual ~Supervised_Cotes_De_Beaune();
 };

 Supervised_Cotes_De_Beaune::Supervised_Cotes_De_Beaune()
{cout << " Supervised_Cotes_De_BeauneConstruction " << endl;}
 Supervised_Cotes_De_Beaune::~Supervised_Cotes_De_Beaune()
{cout << " ~Supervised_Cotes_De_Beaune ";}



main()
{
  cout << " ------------------------ " << endl;
  SupervisedBurgundy instanceOfSupervisedBurgundy;
  cout << endl;

  char stop; cin >> stop;

  cout << " ------------------------ " << endl;
```

```
  Supervised_Cotes_De_Beaune instanceOfSupervised_Cotes_De_Beaune;
  cout << endl;
}

/*
 -------------------------
 WineConstruction
 Wine_InventionConstruction
 How many vine are mixed ? 4
vine 0 ? 1
vine 1 ? 2
vine 2 ? 3
vine 3 ? 4
 Supervised_WineConstruction
vine 01
vine 12
vine 23
vine 34
 SupervisedBurgundyConstruction

f
 -------------------------
 WineConstruction
 Supervised_WineConstruction
vine 0-1073743459
vine 10
vine 2-1073743453
vine 3-1073743428
vine 4-1073743415
vine 5-1073743390
vine 6-1073743374
....
*/
```

```
#include<iostream>
using namespace std;

class Vin{
 private:
        string couleur;
        string nom;
 public:
 Vin(){cout << "Vin " << endl;}
 virtual ~Vin(){}
 virtual void copieCaracteristiques(string&)const{cout <<  "copie
caracteristiques " << endl;}
};

class Vin_Invention : virtual public Vin{
 public:
 Vin_Invention(){cout << "Vin_Invention " ;saisieCaracteristiques(cin);}
 virtual ~Vin_Invention(){}
 virtual void saisieCaracteristiques(istream& is){cout <<  "saisie
caracteristiques " << endl;}
};

class Vin_Supervise : virtual public Vin{
 private:
        static string Historique;
 public:
 Vin_Supervise(){cout << "Vin_Supervise "
;copieCaracteristiques(Historique);}
 virtual ~Vin_Supervise(){}
};

string Vin_Supervise::Historique="-";

 class Bourgogne
   : virtual public Vin_Invention, virtual public Vin_Supervise
 {
 public:
 Bourgogne(){cout << "Bourgogne " << endl;}
 virtual ~Bourgogne(){}
 };

 class Cotes_De_Beaune
   : virtual public Vin_Supervise, virtual public Bourgogne
 {
 public:
 Cotes_De_Beaune(){cout << "Cotes_De_Beaune " << endl;}
 virtual ~Cotes_De_Beaune(){}
 };

int main()
{
 cout << " ------------------------ " << endl;
  Bourgogne instanceOfBourgogne;
  cout << endl;

  cout << " ------------------------ " << endl;
  Cotes_De_Beaune instanceOfCotes_De_Beaune;
  cout << endl;
```

```
}
```

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;

class Vetement{};
class Vaisselle{};

template <typename T>
class Armoire{
private:
        vector<T*> contenu;
public:
        Armoire();
        virtual bool contient(T* v);
};

template <typename T>
Armoire<T>::Armoire(){}

template <typename T>
bool Armoire<T>::contient(T* v){return
find(contenu.begin(),contenu.end(),v)!=contenu.end();}

int main()
{
Vaisselle v;
Armoire<Vaisselle> a;
cout << a.contient(&v)<< endl;
}
```