



POO – Langage C++

Les templates (la généricité)

1^{ère} année ingénieur informatique

Mme Wiem Yaiche Elleuch

2018 - 2019

1. Les patrons/templates de fonctions

2. Les patrons/templates de classes

définition

- **Modèle** que le compilateur utilise pour créer des fonctions au besoin.

```
void permuter( int & a, int & b)
{
    int temp = a;
    a = b;
    b = temp;
}
```

```
void permuter( float & a, float & b)
{
    float temp = a;
    a = b;
    b = temp;
}
```

```
void permuter( char & a, char & b)
{
    char temp = a;
    a = b;
    b = temp;
}
```

Même code
Seul le type est différent



Solution: définir un
patron "template"
pour cette fonction.

```
template <class T>
void permuter(T& a, T& b)
{
    T temp = a;
    a = b;
    b = temp;
}
```



Le compilateur **instancie** ce patron selon
le type indiqué

Cette fonction est vue comme la surdéfinition de fonctions, sauf que nous sommes en présence d'une seule définition qui va être valable pour tous les types: int, double, char etc

Jargon

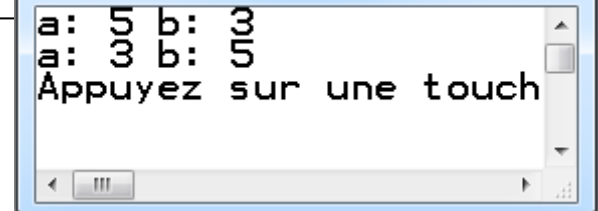
- Les appellations suivantes sont équivalentes
 - Patron de fonction
 - modèle de fonction
 - template de fonction
 - fonction générique

exemple

Instanciation de ce template pour les types int et point

```
template<class T>
void permuter(T&a, T&b)
{
    T aide;
    aide=a;
    a=b;
    b=aide;
}
```

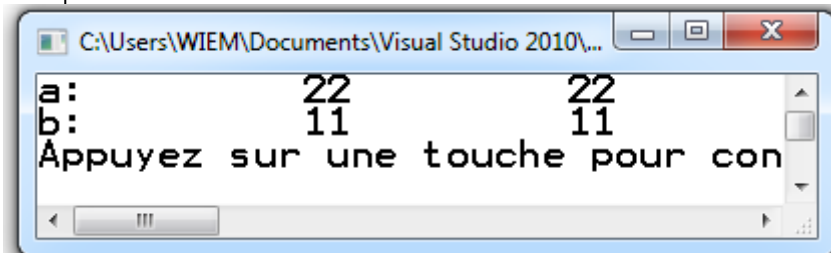
```
void main()
{
    int a=5, b=3;
    cout<<"a: "<<a<<" b: "<<b<<endl;
    permuter<int>(a,b);
    cout<<"a: "<<a<<" b: "<<b<<endl;
    system("PAUSE");
}
```



```
a: 5 b: 3
a: 3 b: 5
Appuyez sur une touche
```

```
void main()
{
    point a(11,11), b(22,22);
    permuter<point>(a,b);

    cout<<"a: "<<a<<endl;
    cout<<"b: "<<b<<endl;
    system("PAUSE");
}
```



```
a: 22 22
b: 11 11
Appuyez sur une touche pour con
```

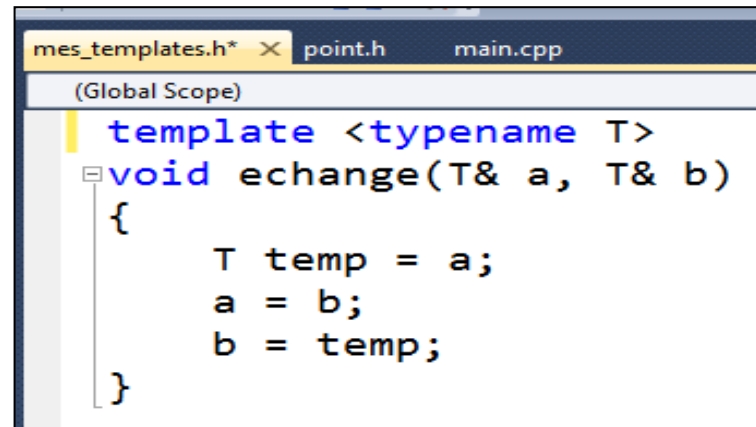
remarque

- `template <class T>` => la fonction est paramétrée par le type T
- T est appelé: paramètre de type.
- Cette fonction est vue comme la surdéfinition de fonctions, sauf que nous sommes en présence d'une seule définition qui va être valable pour tous les types: int, double, char etc
- Un Patron de fonction peut comporter 1 ou plusieurs **paramètres de type**

```
template <class T, class U>  
void fonction (T a, T* b, U c)  
{ .....  
}
```

remarque

- Dans la définition d'un patron, on utilise le mot clé **class** pour indiquer en fait un **type quelconque**:
 - Type prédéfini: short, char, double, int*, char*, int **, etc
 - Type défini par l'utilisateur (structure, classe).
- La norme a introduit le mot clé **typename** qui peut remplacer le mot **class** dans la définition



The screenshot shows a code editor with three tabs: 'mes_templates.h*', 'point.h', and 'main.cpp'. The active tab is 'mes_templates.h*'. The code is in the 'Global Scope' and defines a template function 'exchange' that takes two references of type 'T' and swaps their values using a temporary variable of type 'T'. The keyword 'typename' is used in the template parameter list.

```
template <typename T>
void exchange(T& a, T& b)
{
    T temp = a;
    a = b;
    b = temp;
}
```

Remarque

```
int a=5, b=3;  
permuter<int> (a,b);
```

➔ Le compilateur instancie « T » à « int » et génère la fonction
void permuter (int&,int&).

```
point a(11,11), b(22,22);  
permuter<point> (a,b);
```

➔ Le compilateur instancie « T » à « point » et génère la fonction
void permuter(point &, point&).

Remarque

- D'une manière générale, il est nécessaire que **chaque paramètre de type** apparaisse **au moins une fois** dans l'**entête** du patron.
- En général, les définitions de patrons de fonctions figureront dans des fichiers d'extension .h

Définition d'un patron de fonctions

- les paramètres de type peuvent être employés n'importe où un type effectif est permis.
 - Dans l'en tête
 - Dans les déclarations de variables locales
 - Dans les instructions (sizeof, new, etc)

```
template <class T, class U>
void fct (T a, T * b, U c)
{
    T x ; // variable locale x de type T
    U * adr; // variable locale adr de type U *
    ...
    adr = new T[10]; // allocation tableau de 10 éléments de type T
    ...
    n=sizeof(T); // instruction utilisant le type T
}
```

Avantage et inconvénient des templates de fonctions

Avantage:

- il suffit d'écrire une seule fois la définition d'une fonction pour que le compilateur puisse automatiquement l'adapter à n'importe quel type

Inconvénient:

- toutes les fonctions ainsi fabriquées par le compilateur doivent correspondre à la même définition, donc au même algorithme.

Surdéfinition de fonctions

- un patron de fonctions peut être surdéfini.

```
Template<class T>
T minimum (T a, T b) // 2 arguments
{
    if (a<b) return a;
    else return b;
}
```

```
Template<class T>
T minimum (T a, T b, T c) // 3 arguments
{
    return minimum ( minimum(a,b) , c);
}
```

Spécialisation de fonctions de patron

- Un patron de fonctions peut être **spécialisé**

```
Template<class T>
T minimum (T a, T b)
{
    if (a<b) return a;
    else return b;
}
```

Si T est instancié au type char*
a<b → comparaison des valeurs des pointeurs a
et b (adresses) et non pas une comparaison
lexicographique des chaînes

- Solution:** C++ autorise à fournir, **en plus** de la définition d'un patron, la définition d'une ou de plusieurs fonctions pour certains types d'arguments.

```
Template<class T>
char* minimum (char* a, char* b)
{
    if (strcmp(a,b) < 0) return a;
    else return b;
}
```

1. Les patrons/templates de fonctions

2. Les patrons/templates de classes

Généralités

- **Modèle** que le compilateur utilise pour créer des classes au besoin.
- il suffira d'écrire une seule fois la définition de la classe pour que le compilateur puisse automatiquement l'adapter à différents types.
- Le mot **class** peut être remplacé par **typename**.

```
template<class T>
class point
{
    T x;
    T y;
public:
    point(T =0, T =0);
    void afficher();
};
```

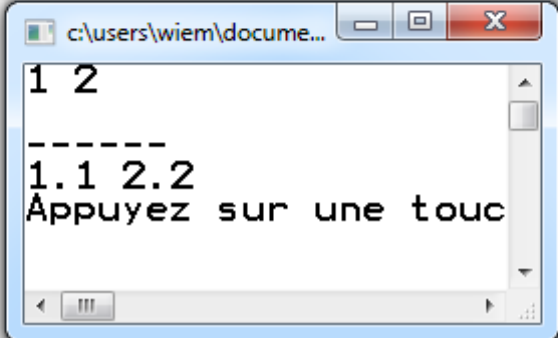
la mention **template <class T>** précise que l'on a affaire à un patron (template) dans lequel apparaît un paramètre de type nommé **T**;

Exemple classe template point

Les fonctions membres sont déclarés **inlines** (à l'intérieur de la classe)

```
point.h  X  main.cpp*
point<T>
template<class T>
class point
{
    T x;
    T y;
public:
    // fonction membres inlines
    point(T abs=0, T ord=0)
    {
        x=abs;
        y=ord;
    }
    void afficher()
    {
        cout<<x<<" "<<y<<endl;
    }
};
```

```
point.h  main.cpp* X
(Global Scope)
#include<iostream>
using namespace std;
#include"mes_templates.h"
#include"point.h"
void main()
{
    point<int> a(1,2);
    a.afficher();
    cout<<"\n-----"<<endl;
    point<float> b(1.1,2.2);
    b.afficher();
    system("PAUSE");
}
```

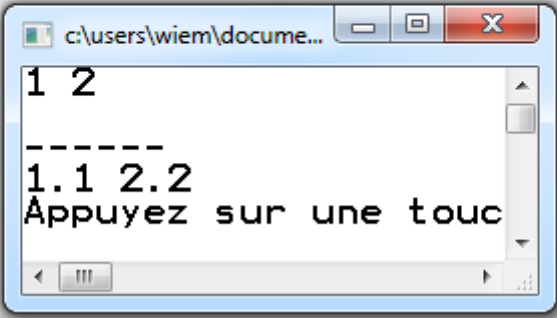


Exemple classe template point

Les fonctions membres sont déclarés à l'extérieur de la classe

```
point.h* x main.cpp
point<T>
template<class T>
class point
{
    T x;
    T y;
public:
    point(T =0, T =0);
    void afficher();
};
// définition des fonctions à
// l'extérieur de la classe
template<class T>
point<T>::point(T abs=0, T ord=0)
{
    x=abs;
    y=ord;
}
template<class T>
void point<T>::afficher()
{
    cout<<x<<" "<<y<<endl;
}
```

```
point.h main.cpp x
(Global Scope)
#include<iostream>
using namespace std;
#include"mes_templates.h"
#include"point.h"
void main()
{
    point<int> a(1,2);
    a.afficher();
    cout<<"\n-----"<<endl;
    point<float> b(1.1,2.2);
    b.afficher();
    system("PAUSE");
}
```



```
c:\users\wiem\docume...
1 2
-----
1.1 2.2
Appuyez sur une touche
```

Définition d'un patron de classes

- Si une fonction membre est définie à l'extérieur de la définition du patron, il faut rappeler au compilateur la liste de paramètres (template) et préfixer l'en-tête de la fonction membre du **nom du patron** accompagné de ses **paramètres**.
- ➔ redondance constatée, mais non justifiée, par le fondateur du langage lui-même, Stroustrup.)

```
template <class T, class U>
class exemple
{
    void afficher();
};
```

```
template <class T, class U>
void exemple<T,U>::afficher()
{
    // les instructions d'affichage
};
```

Les paramètres de type d'un patron de classes

- Les paramètres de type peuvent être en nombre quelconque et utilisés dans la définition du patron de classes

```
template<class T, class U, class V>
class essai
{
    T x;    // membre x de type T
    U tab[5]; // tableau de 5 éléments de type U
    ...
    V fct (int, U); // fonction membre qui renvoie un
    // résultat de type V, et ayant deux arguments de type int et U
    ...
};
```

- Exemples d'instances de ce patron

```
essai<int , float , char*> e1;
essai< point<int>, point<float> , double> e2;
```

Cas des fonctions amies

```
mes_templates.h* x main.cpp
(Global Scope)
template<class T>
class point
{
    T x,y;
public:
    point(T dx=0, T dy=0);
    void afficher();
    friend ostream& operator<< <>(ostream&,point&);
};

template<class T>
ostream& operator<< (ostream& out, point<T>& pt)
{
    out<<pt.x<<" "<<pt.y<<endl;
    return out;
}
```

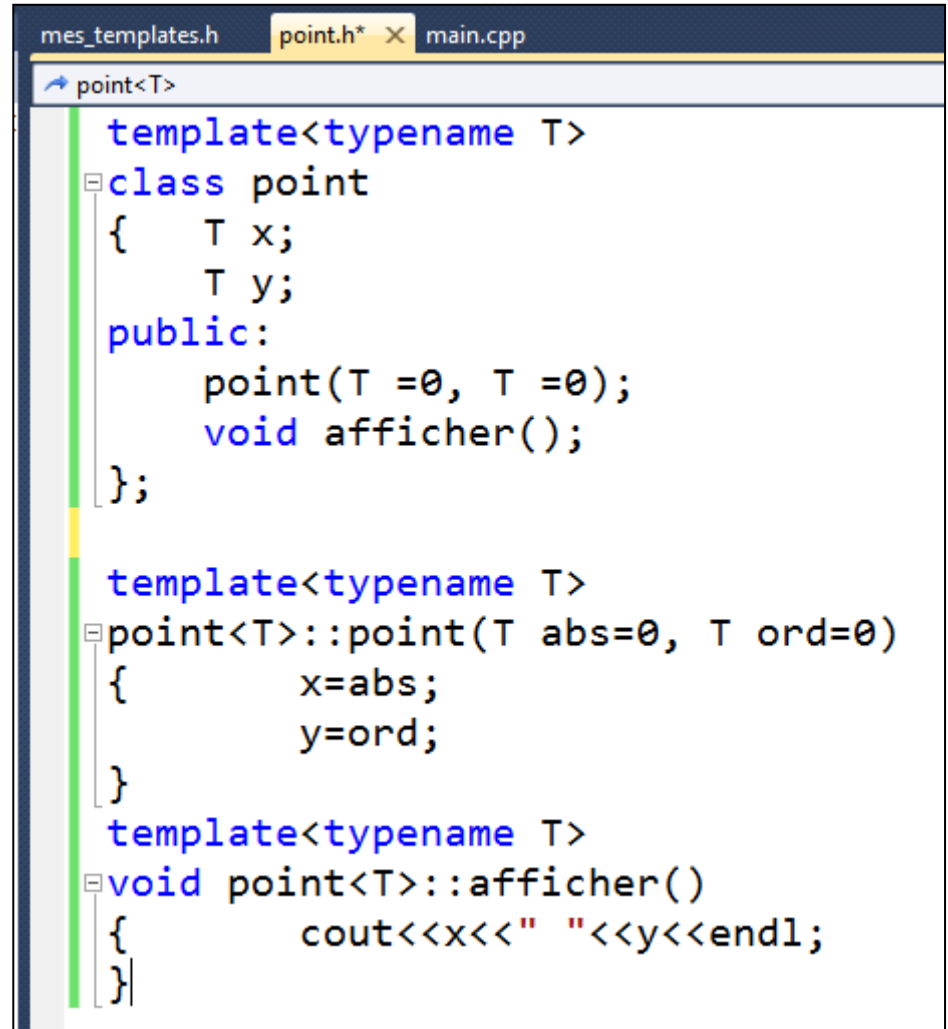
Ne pas oublier <>

```
void main()
{
    point<int> a(12,13);
    cout<<a;
    cout<<"\n -----"<<endl;
    point<float> b(44.44,55.55);
    cout<<b;
    system("PAUSE");
}
```

```
c:\users\wiem\documents\visual studio 2010\Projects\AA_S18_templates...
12 13
-----
44.44 55.55
Appuyez sur une touche pour continuer...
```

Class ou typename

- La norme a introduit le mot clé **typename** qui peut remplacer le mot **class** dans la définition

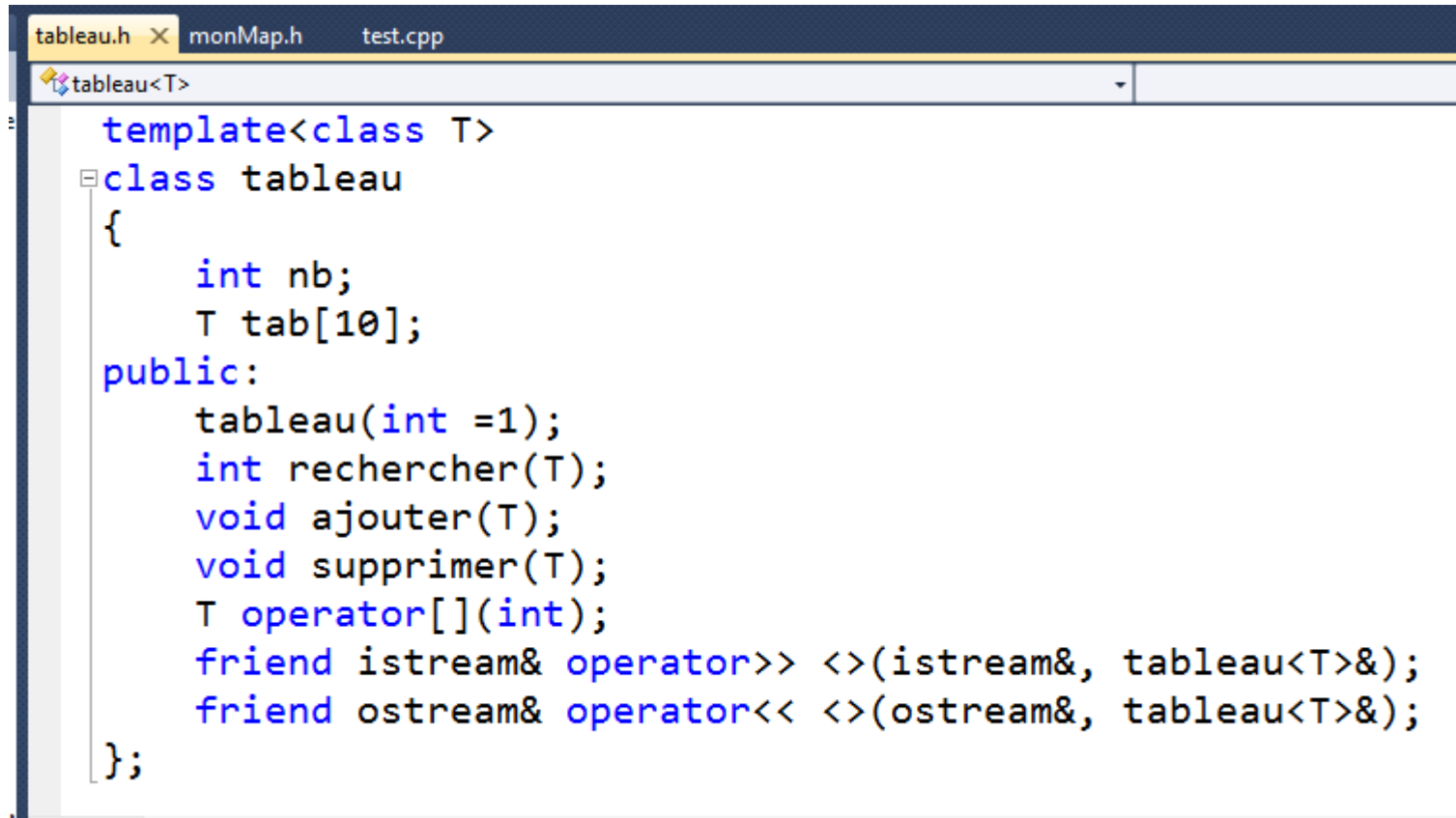


```
mes_templates.h  point.h*  main.cpp
point<T>
template<typename T>
class point
{
    T x;
    T y;
public:
    point(T =0, T =0);
    void afficher();
};

template<typename T>
point<T>::point(T abs=0, T ord=0)
{
    x=abs;
    y=ord;
}

template<typename T>
void point<T>::afficher()
{
    cout<<x<<" "<<y<<endl;
}
```

Exemple template classe tableau



The screenshot shows a code editor with three tabs: 'tableau.h', 'monMap.h', and 'test.cpp'. The 'tableau.h' tab is active, showing the following C++ code:

```
template<class T>
class tableau
{
    int nb;
    T tab[10];
public:
    tableau(int =1);
    int rechercher(T);
    void ajouter(T);
    void supprimer(T);
    T operator[](int);
    friend istream& operator>> <>(istream&, tableau<T>&);
    friend ostream& operator<< <>(ostream&, tableau<T>&);
};
```

```

template<class T>
T tableau<T>::operator[](int i)
{
    return tab[i];
}

template<class T>
void tableau<T>::supprimer(T a)
{
    int res=rechercher(a);
    if(res!=-1)
    {
        for(int i=res; i<nb-1; i++)
            tab[i]=tab[i+1];
        nb--;
    }
    else cout<<"\n element n'existe pas "<<endl;
}

```

```

template<class T>
void tableau<T>::ajouter(T a)
{
    if(nb<10 && (rechercher(a)==-1))
    {
        tab[nb]=a;
        nb++;
    }
}

template<class T>
int tableau<T>::rechercher(T a)
{
    for(int i=0; i<nb;i++)
        if(tab[i]==a) return i;
    return -1;
}

```

```
tableau.h* X monMap.h test.cpp
tableau<T>
rechercher(T a)

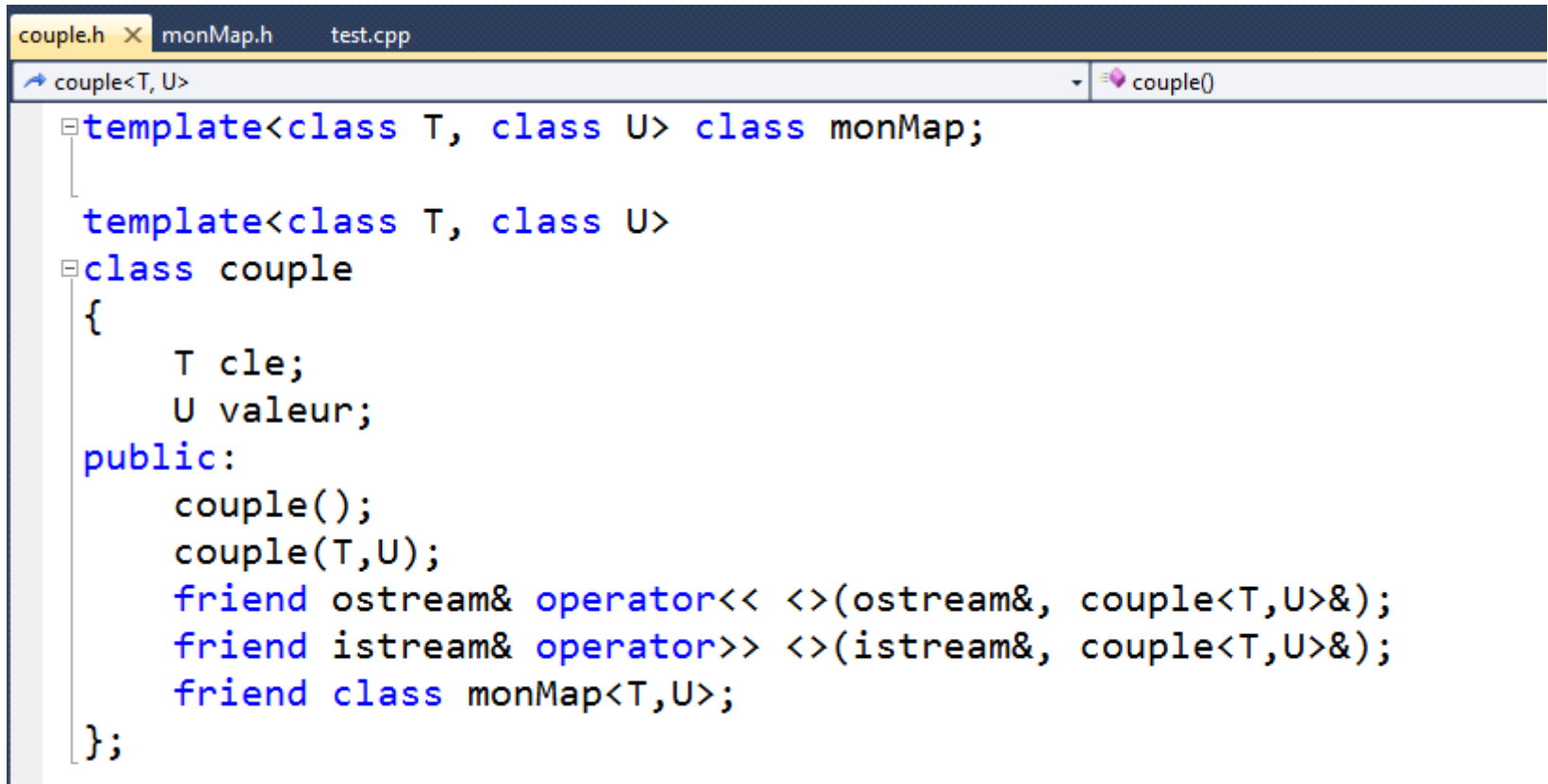
template<class T>
tableau<T>::tableau(int n)
{
    nb=n;
}

template<class T>
istream& operator>> (istream& in, tableau<T>& t)
{
    cout<<"\n remplissage "<<endl;
    for(int i=0; i<t.nb;i++)
        in>>t.tab[i];
    return in;
}
```

```
tableau.h* X monMap.h test.cpp
tableau<T>
rechercher(T a)

template<class T>
ostream& operator<< (ostream& out, tableau<T>& t)
{
    out<<"\n nbre d'elements "<<t.nb<<endl;
    for(int i=0; i<t.nb;i++)
    {
        out<<t.tab[i]<<" ";
        out<<endl;
    }
    return out;
}
```


Exemple template classe couple



The screenshot shows a C++ IDE with three tabs: 'couple.h', 'monMap.h', and 'test.cpp'. The 'couple.h' tab is active, displaying the following code:

```
couple.h  X  monMap.h  test.cpp
couple<T, U>  couple()

template<class T, class U> class monMap;

template<class T, class U>
class couple
{
    T cle;
    U valeur;
public:
    couple();
    couple(T,U);
    friend ostream& operator<< <>(ostream&, couple<T,U>&);
    friend istream& operator>> <>(istream&, couple<T,U>&);
    friend class monMap<T,U>;
};
```

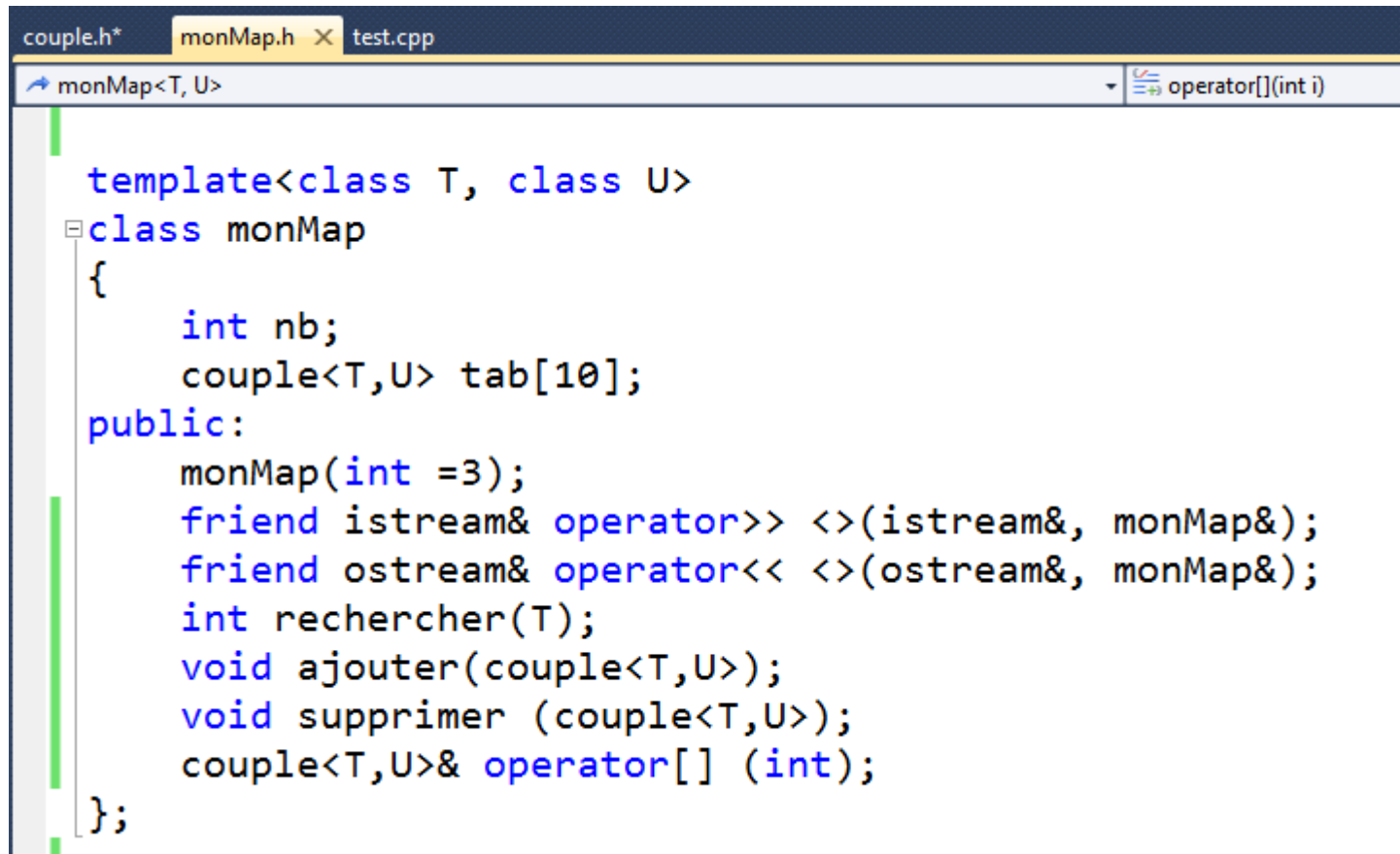
```
template<class T, class U>
couple<T,U>::couple()
{
}

template<class T, class U>
couple<T,U>::couple(T c, U v)
{
    cle=c;
    valeur=v;
}

template<class T, class U>
ostream& operator<< (ostream& out, couple<T,U>& c)
{
    out<<"\n cle "<<c.cle<<"  valeur "<<c.valeur<<endl;
    return out;
}
```

```
template<class T, class U>
istream& operator>> (istream& in, couple<T,U>& c)
{
    in>>c.cle>>c.valeur;
    return in;
}
```

Exemple template classe monMap



The screenshot shows a C++ IDE with three tabs: couple.h*, monMap.h (active), and test.cpp. The active tab displays the definition of a template class monMap. The code is as follows:

```
template<class T, class U>
class monMap
{
    int nb;
    couple<T,U> tab[10];
public:
    monMap(int =3);
    friend istream& operator>> <>(istream&, monMap&);
    friend ostream& operator<< <>(ostream&, monMap&);
    int rechercher(T);
    void ajouter(couple<T,U>);
    void supprimer (couple<T,U>);
    couple<T,U>& operator[] (int);
};
```

(Global Scope)

```
template<class T, class U>
couple<T,U>& monMap<T,U>::operator[](int i)
{
    return tab[i];
}
```

```
template<class T, class U>
void monMap<T,U>::supprimer(couple<T,U> c)
{
    int res=rechercher(c.cle);
    if(res!=-1)
    {
        for(int i=res; i<nb;i++)
            tab[i]=tab[i+1];
        nb--;
    }
}
```

```
template<class T, class U>
void monMap<T,U>::ajouter(couple<T,U> c)
{
    if (nb<10 && rechercher(c.cle)==-1)
    {
        tab[nb]=c;
        nb++;
    }
}
```

```
template<class T, class U>
int monMap<T,U>::rechercher(T c)
{
    for(int i=0; i<nb; i++)
        if(tab[i].cle==c) return i;
    return -1;
}
```

```
template<class T, class U>
monMap<T,U>::monMap(int n)
{
    nb=n;
}
template<class T, class U>
istream& operator>> (istream& in, monMap<T,U>& m)
{
    cout<<"\n remplissage "<<endl;
    for(int i=0; i<m.nb;i++)
        in>>m.tab[i];
    return in;
}
template<class T, class U>
ostream& operator<< (ostream& out, monMap<T,U>& m)
{
    out<<"\n affichage "<<endl;
    for(int i=0; i<m.nb;i++)
    {
        out<<m.tab[i]<<" ";
        out<<endl;
    }
    return out;
}
```

(Global Scope)

```
void main()
{
    monMap<int,point> m;
    cin>>m;
    cout<<m;
    cout<<"\n-----"<<endl;
    point a(33,33);
    couple<int,point> c(789,a);
    m.ajouter(c);

    cout<<"\n-----"<<endl;
    cout<<m;
    system("PAUSE");
}
```

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\testPOO\Debug...

```
123
11
11
456
22
22

le nbre d'elements 2
123      11      11
456      22      22

-----

le nbre d'elements 3
123      11      11
456      22      22
789      33      33

Appuyez sur une touche pour continuer...
```

```
monMap.h    couple.h    main.cpp X
(Global Scope)

void main()
{
    monMap<int,point> m(3);
    cin>>m;
    cout<<m;
    cout<<"\n-----"<<endl;
    couple<int,point> c(456, point(22,22));
    m.supprimer(c);

    cout<<"\n-----"<<endl;
    cout<<m;
    system("PAUSE");
}
```

```
C:\Users\WIEM\Documents\Visual Studio 2010\Projects\test...
123
11
11
456
22
22
789
33
33

le nbre d'elements 3
123      11      11
456      22      22
789      33      33

-----

le nbre d'elements 2
123      11      11
789      33      33

Appuyez sur une touche pour continuer...
```

Exemple template class pile

```
pile.h* x test.cpp
(Global Scope)
struct element
{
    T info;
    element*suisvant;
};
template<class T>
class pile
{
    element<T>*somet;
public:
    pile();
    ~pile();
    void empiler(T);
    T depiler();
    friend ostream& operator<< <>(ostream&, pile<T>&);
    friend istream& operator>> <>(istream&, pile<T>&);
    bool estVide();
    int nbElement();
    void afficher(); // afficher sans depiler pour les tests
    pile(const pile<T>&);
    pile& operator=(pile<T>&);
};
```


Exemple template class pile

```
pile.h* X test.cpp
(Global Scope)

template<class T>
void pile<T>::empiler(T d)
{
    element<T> *e=new element<T>;
    e->suivant=sommet;
    e->info=d;
    sommet=e;
}

template<class T>
T pile<T>::depiler()
{
    if(sommet !=NULL)
    {
        element<T>*e=sommet;
        T data=e->info;
        sommet=sommet->suivant;
        delete e;
        return data;
    }
}
```

```
template<class T>
void pile<T>::afficher()
{
    if(sommet==NULL) cout<<"\n pile vide "<<endl;

    else
    {
        element<T>*e=sommet;
        while(e!=NULL)
        {
            cout<<e<<" "<<e->info<<endl;
            e=e->suivant;
        }
    }
}
```

Exemple template class pile

```
pile.h* X test.cpp
(Global Scope)
template<class T>
ostream& operator<< (ostream& out, pile<T>& p)
{
    while(p.sommet!=NULL)
        out<<p.depiler()<<endl;
    return out;
}

template<class T>
istream& operator>> (istream& in, pile<T>& p)
{
    char rep;
    do
    {
        T data;
        cin>>data;
        p.empiler(data);
        cout<<"\n rajouter ?"<<endl;
        cin>>rep;
    }
    while(rep=='o' || rep=='O');

    return in;
}
```

```
pile.h* X test.cpp
pile<T>
template<class T>
int pile<T>::nbElement()
{
    element<T>*e=sommet;
    int nb=0;
    while(e!=NULL)
    {
        e=e->suivant;
        nb++;
    }
    return nb;
}

template<class T>
bool pile<T>::estVide()
{
    return (sommet==NULL);
}

template<class T>
pile<T>::pile()
{
    sommet=NULL;
}
```

Exemple template class pile

```
pile.h* X test.cpp
pile<T>
template<class T>
pile<T>::~~pile()
{
    while(sommet!=NULL)
        depiler();
}

template<class T>
pile<T>::pile(const pile<T>& w)
{
    sommet=NULL;
    element<T> *e, *courant, *tmp=NULL;
    courant=w.sommet;
    while(courant!=NULL)
    {
        e=new element<T>(*courant);

        if(sommet==NULL)sommet=e;
        else tmp->suivant=e;

        tmp=e;
        courant=courant->suivant;
    }
}
```

```
pile.h* X test.cpp
pile<T>
template<class T>
pile<T>& pile<T>::operator= (pile<T>& w)
{
    if(this!=&w)
    {
        while(sommet!=NULL)
            depiler();

        sommet=NULL;
        element<T> *e, *courant, *tmp=NULL;
        courant=w.sommet;
        while(courant!=NULL)
        {
            e=new element<T>(*courant);

            if(sommet==NULL)sommet=e;
            else tmp->suivant=e;

            tmp=e;
            courant=courant->suivant;
        }

        return *this;
    }
}
```

Templates de classes et héritage

- Il est très facile de combiner la notion d'héritage avec celle de patron de classes:
 - Classe "ordinaire" dérivée d'une classe patron
 - Patron de classes dérivé d'une classe "ordinaire".
 - Patron de classes dérivé d'un patron de classes

Classe "ordinaire" dérivée d'une classe patron

- Classe "ordinaire" dérivée d'une classe patron (c'est-a-dire d'une instance particulière d'un patron de classes).
- Par exemple, si A est une classe patron définie par *template <class T>A*:
class B : public A <int>
// B dérive de la classe patron A<int>
- on obtient une seule classe nommée B.

```
pointtt.h x pointColore_int.h pointColore_int.cpp main.cpp
pointtt<T>
template<class T>
class pointtt
{
    T x;
    T y;
public:
    pointtt(T =0, T =0);
    virtual void afficher();
    friend ostream& operator<< <> (ostream&, pointtt<T>&);
    friend istream& operator>> <> (istream&, pointtt<T>&);
};
```

```
pointtt.h pointColore_int.h* x pointColore_int.cpp main.cpp
(Global Scope)
#include"pointtt.h"
class pointColore_int: public pointtt<int>
{
    int couleur;
public:
    pointColore_int(int =11, int =22, int =33);
    ~pointColore_int(void);
    void afficher(string = "");
};
```

```
pointt.h  pointColore_int.h*  pointColore_int.cpp*  main.cpp
(Global Scope)

#include "pointColore_int.h"
pointColore_int::pointColore_int(int abs, int ord, int coul):
    pointt<int>(abs,ord), couleur(coul)
{
}
pointColore_int::~~pointColore_int(void)
{
}
void pointColore_int::afficher(string msg)
{
    cout<<msg<<endl;
    pointt<int>::afficher();
    cout<<"\n couleur " <<couleur<<endl;
}
```

```
pointt.h  pointColore_int.h*  pointColore_int.cpp*  main.cpp
(Global Scope)

#include "pointColore_int.h"
void main()
{
    pointColore_int a;
    a.afficher();
    system("PAUSE");
}
```

```
C:\Users\WIEM\Documents\Visual Studio 2010\Proj...
11  22
    couleur 33
Appuyez sur une touche pour continuer..
```

Patron de classes dérivé d'une classe "ordinaire".

- Par exemple, A étant une classe ordinaire :
template <class T> class B : public A
- on obtient une famille de classes (de paramètre de type T).
- L'aspect "patron" a été introduit ici au moment de la dérivation.

Patron de classes dérivé d'un patron de classes

- Cette possibilité peut revêtir deux aspects selon que l'on introduit ou non de nouveaux paramètres lors de la dérivation.
- Par exemple, si A est une classe patron définie par ***template <class T> A***, on peut :

- définir une nouvelle famille de fonctions dérivées par:

template <class T> class B : public A <T>

Dans ce cas, il existe autant de classes dérivées possibles que de classes de base possibles

- définir une nouvelle famille de fonctions dérivées par :

template <class T, class U> class B : public A <T>

Dans ce cas, on peut dire que chaque classe de base possible peut engendrer une famille de classes dérivées (de paramètre de type U).

Dérivation de patrons avec les mêmes paramètres

- A partir du patron *templale* `<class T> class point`, nous dérivons un patron nommé *pointCol* dans lequel le nouveau membre introduit est du même type T que les coordonnées du point:

```
pointColl.h  pointt.h  pointColore_int.h  pointColore_int.cpp  main
pointColl<T>

template<class T>
class pointColl: public pointt<T>
{
    T couleur;
public:
    pointColl(T , T , T );
    void afficher(string = "");
};
```

```
pointColl<T>  affi

template<class T>
pointColl<T>::pointColl(T abs, T ord, T coul):
    pointt<T>(abs,ord), couleur(coul)
{
}

template<class T>
void pointColl<T>::afficher(string msg)
{
    cout<<msg<<endl;
    pointt<T>::afficher();
    cout<<"\n couleur " <<couleur<<endl;
}
```

```
pointColl.h  pointt.h  pointColore_int.h  pointColore_int.cpp  main.cpp
(Global Scope)

#include"pointColl.h"

void main()
{
    pointColl<int> a(11,22,33);
    a.afficher();
    system("PAUSE");
}
```

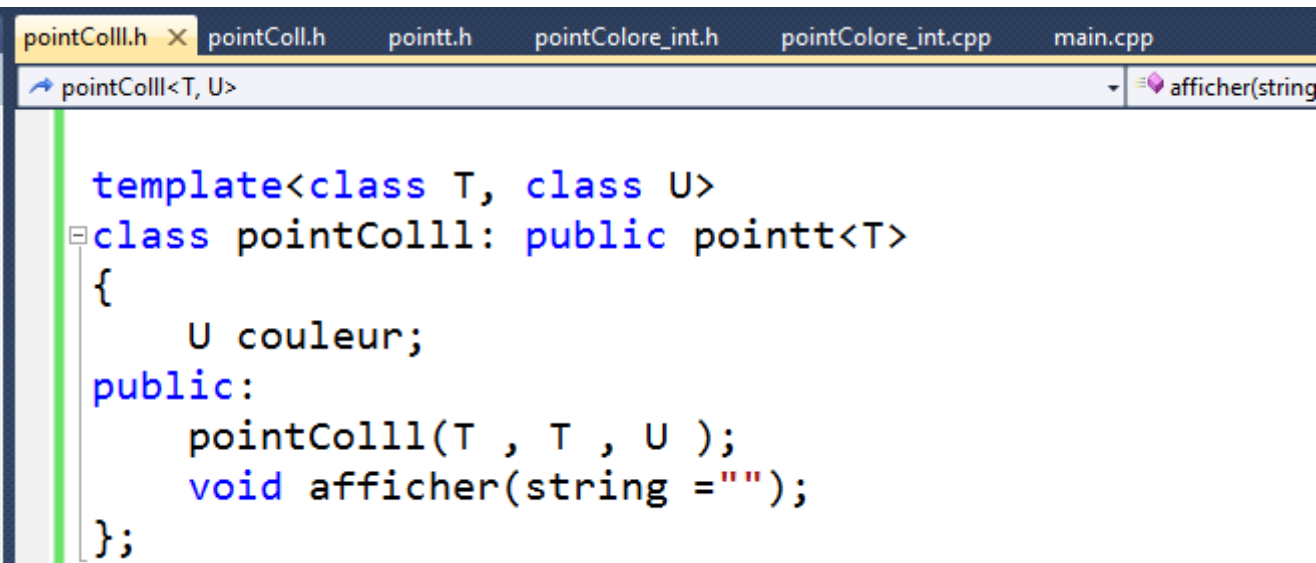
```
Sélectionner C:\Users\WIEM\Documents\Visual Stu...

11  22

couleur 33
Appuyez sur une touche pour continuer..
```

Dérivation de patrons avec introduction d'un nouveau paramètre

- A partir du patron *template* `<class T> class point`, nous dérivons un patron nommé *pointcoll* dans lequel le nouveau membre introduit est d'un type `U` différent de celui des coordonnées du point:



The screenshot shows a C++ IDE with several tabs: `pointColl.h` (active), `pointColl.h`, `pointt.h`, `pointColore_int.h`, `pointColore_int.cpp`, and `main.cpp`. The active tab `pointColl.h` displays the following code:

```
template<class T, class U>
class pointColl: public pointt<T>
{
    U couleur;
public:
    pointColl(T , T , U );
    void afficher(string = "");
};
```

```
template<class T, class U>
pointColl<T,U>::pointColl(T abs, T ord, U coul):
    pointt<T>(abs,ord), couleur(coul)
{
}
template<class T, class U>
void pointColl<T,U>::afficher(string msg)
{
    cout<<msg<<endl;
    pointt<T>::afficher();
    cout<<"\n couleur "<<coul<<endl;
}
```

(Global Scope)

```
#include"pointColl.h"
void main()
{
    pointColl<int, float> a(11,22,33.33);
    a.afficher();
    system("PAUSE");
}
```

C:\Users\WIEM\Documents\Visual Studio 2010\Projects\t...

```
11  22
    couleur 33.33
Appuyez sur une touche pour continuer...
```