

La surcharge d'opérateurs POO - cours 2

Le point abordé dans ce cours est le suivant :

- La surcharge d'opérateurs (surdéfinition)

Ce concept permet de redéfinir le sens premier d'un opérateur du langage (+, -, =, ++, --, (), [], ->, ...).

Ce concept peut être utilisé dans le cadre d'une classe à travers les fonctions membres.

La surcharge d'opérateur utilise le mot clé « operator » suivie de l'opérateur du langage à redéfinir soit la syntaxe suivante :

```
<Type de retour> operator <op> (<paramètres>) ; // entête dans la classe  
<Type de retour> <classe> ::operator <op> (paramètres) ; // implémentation
```

Un opérateur peut bien sûr être surchargé plusieurs fois.

Reprenons l'exemple développé dans le TD1 basé sur l'utilisation des vecteur3d.

Il est possible d'implémenter dans la classe vecteur3d des opérateurs permettant d'additionner des vecteurs ou de les multiplier. On peut aussi redéfinir des opérateurs qui permettront la comparaison des vecteurs.

Rappels :

Une fonction d'addition (addition) permettra d'obtenir la somme de deux vecteurs, une autre (scalaire) permettra d'obtenir le produit scalaire.

$$\vec{A} + \vec{B} = (Ax + Bx, Ay + By, Az + Bz)$$

$$\vec{A} \cdot \vec{B} = (Ax \cdot Bx + Ay \cdot By + Az \cdot Bz)$$

Les opérateurs == et != permettront de tester la coïncidence ou non des deux vecteurs.

<< implementation >> + vecteur3d	
• - floatX	
• - floatY	
• - floatZ	
+ vecteur3d	<< create >>
+ ~vecteur3d	<< destroy >>
+ ==	
+ !=	
+ operator_add	
+ operator_multi	

1. Interface et implémentation

```
#ifndef _VECTEUR3D_H
#define _VECTEUR3D_H

#include <iostream>
using namespace std;

class vecteur3d
{
    private :

        float floatX;
        float floatY;
        float floatZ;

    public :

        vecteur3d(float floatX = 0, float floatY = 0, float floatZ =
0);
        vecteur3d operator +(vecteur3d &V1);
        float operator *(vecteur3d &V1);
        bool operator ==(vecteur3d &V1);
        bool operator !=(vecteur3d &V1);
        ~vecteur3d();
};

#endif _VECTEUR3D_H
```

```
#ifndef _VECTEUR3D_CPP
#define _VECTEUR3D_CPP

#include "vecteur3d.h"

vecteur3d::vecteur3d(float floatX, float floatY, float floatZ)
{
    this->floatX = floatX;
    this->floatY = floatY;
    this->floatZ = floatZ;
}

vecteur3d vecteur3d::operator +(vecteur3d &V1)
{
    vecteur3d temp;
    temp.floatX = this->floatX + V1.floatX;
    temp.floatY = this->floatY + V1.floatY;
    temp.floatZ = this->floatZ + V1.floatZ;
    return(temp);
}

float vecteur3d::operator *(vecteur3d &V1)
{
    return(this->floatX * V1.floatX + this->floatY * V1.floatY + this->floatZ * V1.floatZ);
}

bool vecteur3d::operator ==(vecteur3d &V1)
{

```

```

        if (this->floatX == V1.floatX && this->floatY == V1.floatY && this->floatZ == V1.floatZ )
        {
            return(true);
        }
        else
        {
            return(false);
        }
    }

bool vecteur3d::operator !=(vecteur3d &V1)
{
    if (this->floatX != V1.floatX && this->floatY != V1.floatY && this->floatZ != V1.floatZ )
    {
        return(true);
    }
    else
    {
        return(false);
    }
}

vecteur3d::~vecteur3d()
{
}

#endif _VECTEUR3D_CPP

```

2. Utilisation de la classe

Ecrire un programme d'application de la classe vecteur3d

```

#include "stdafx.h"
#include "vecteur3d.h"

int _tmain(int argc, _TCHAR* argv[])
{
    vecteur3d V1(1,2,3);
    vecteur3d V2(1,2,3);
    vecteur3d V3(1,2,3);
    if ( V1 == V2 )
    {
        cout<<"egalite";
    }
    else
    {
        cout<<"inegalite";
    }
    vecteur3d V4;
    V4 = V2 + V1;
    cout<<"Produit scalaire : "<<V2*V1<<"\n";

    vecteur3d V5(1,1,1);
    cout<<"Produit scalaire : "<<V5*V4<<"\n";
    return 0;
}

egaliteProduit scalaire : 14

```

3. Ma classe chaîne

Maintenant que vous connaissez les fondements de la redéfinition d'opérateurs, vous allez devoir écrire votre propre classe « chaîne ». Bien sûr nous n'allons pas réinventer la roue, mais en guise d'entraînement, vous devrez implémenter quelques méthodes.

Si vous utilisez une des dernières versions de Visual Studio, les vieilles fonctions `strcpy`, `strncpy`, `strcat`, `strncat` sont classées « deprecated ».

Il convient d'utiliser leurs nouvelles définitions `strcpy_s`, `strncpy_s`, `strcat_s`, `strncat_s`.

Pour des raisons de simplification du travail, la gestion de erreurs sur les longueurs de chaînes n'est pas implémentée. Une variante pourra régler le problème. Dans ce cas l'analyse faite est à repenser.

Dans notre exemple, vous devrez surcharger plusieurs fois les opérateurs standards. Attention un opérateur unaire ou binaire reste unaire ou binaire même une fois la surcharge effectuée.

Votre programme devra offrir les fonctionnalités suivantes :

```
int _tmain(int argc, _TCHAR* argv[])
{
    chaine maChaine(20); // création de la chaîne maChaine de longueur 20 caractères
    maChaine << " Hello World! "; // place le mot "Bonjour" dans la chaîne maChaine
    cout << maChaine << "\n"; // affichage du contenu de la chaîne maChaine
    maChaine = "\0"; // efface le contenu de la chaîne maChaine une autre variante possible
    // pour l'affectation
    maChaine = " Hi, hi, hi !"; // place le message dans la chaîne maChaine
    cout << maChaine << "\n"; // affichage du contenu de la chaîne maChaine
    cout << maChaine[1] << "\n"; // affichage du contenu de la position 1 de la chaîne maChaine
    cout << maChaine.longueur() << "\n"; // affichage de la longueur de la chaîne maChaine

    chaine autreChaine(50); // création de la chaîne autreChaine de longueur 50 caractères
    autreChaine = maChaine + maChaine; // concaténation de deux chaînes et affectation du
    // résultat dans autreChaine
    cout << autreChaine << "\n"; // affichage du contenu de la chaîne autreChaine

    chaine uneChaine(80); // création la chaîne uneChaine longueur 80 caractères
    uneChaine = autreChaine + " un ensemble de caracteres!"; // concatenation d'une chaîne
    // et d'un ensemble de caractères
    cout << uneChaine << "\n"; // affichage du contenu de la chaîne uneChaine
    return 0;
}
```

Le programme précédent donne le résultat suivant :

```
Hello World!
Hi, hi, hi !
H
13
Hi, hi, hi ! Hi, hi, hi !
Hi, hi, hi ! Hi, hi, hi ! un ensemble de caracteres!
```

```

#ifndef _CHAINE_H
#define _CHAINE_H

#include <iostream>
using namespace std;

class chaine
{
    private :

        int intLongueur;
        char *lesCaracteres;

    public :

        chaine(int intLenght);
        int longueur();
        void operator << (char * charContenu); // affecter une valeur à
la chaîne
        char operator [] (int intPosition); // editer un caractère de la
chaîne placé à la position intPosition
        chaine operator + (chaine &C); // concaténation de chaîne
        chaine operator + (char * charContenu);
        void operator = (chaine &C); // affectation de chaîne
        void operator = (char * charContenu); // une autre affectation
de chaîne
        friend ostream & operator << (ostream & out, chaine &C) ; //
surcharge de cout pour afficher la chaîne
        ~chaine();

};

#endif _CHAINE_H

```

```

#ifndef _CHAINE_CPP
#define _CHAINE_CPP

#include "chaine.h"

chaine::chaine(int intLenght)
{
    this->lesCaracteres = new char[intLenght+1];
    this->intLongueur = intLenght+1;
    //strncpy(this->lesCaracteres, "\0", intLenght); deprecated now !
    strncpy_s(this->lesCaracteres, this->intLongueur, "\0", this-
>intLongueur);
}

char chaine::operator [] (int intPosition)
{
    return(lesCaracteres[intPosition]);
}

chaine chaine::operator + (chaine &C) // l'objet courant à gauche du signe +
l'objet en paramètre à droite du +
{

```

```

    chaine temp(this->intLongueur + C.intLongueur);
    temp.intLongueur = this->intLongueur + C.intLongueur + 1;
    //strncpy(temp.lesCaracteres,"\0",temp.intLongueur); deprecated now !
    strncpy_s(temp.lesCaracteres,temp.intLongueur,"\0",temp.intLongueur);
    //strncpy(temp.lesCaracteres, C.lesCaracteres, C.intLongueur);
deprecated now !
    //strncat(temp.lesCaracteres, this->lesCaracteres, this-
>intLongueur); deprecated now !
    strncpy_s(temp.lesCaracteres, temp.intLongueur, this->lesCaracteres,
this->intLongueur);
    strncat_s(temp.lesCaracteres, temp.intLongueur, C.lesCaracteres,
C.intLongueur);
    return(temp);
}

chaine chaine::operator +(char * charContenu) // l'objet courant à gauche
du signe + l'objet en paramètre à droite du +
{
    chaine temp(this->intLongueur + (int)strlen(charContenu));
    temp.intLongueur = this->intLongueur + (int)strlen(charContenu) + 1;
    //strncpy(temp.lesCaracteres,"\0",temp.intLongueur); deprecated now !
    strncpy_s(temp.lesCaracteres,temp.intLongueur,"\0",temp.intLongueur);
    //strncpy(temp.lesCaracteres, C.lesCaracteres, C.intLongueur);
deprecated now !
    //strncat(temp.lesCaracteres, this->lesCaracteres, this-
>intLongueur); deprecated now !
    strncpy_s(temp.lesCaracteres, temp.intLongueur, this->lesCaracteres,
this->intLongueur);
    strncat_s(temp.lesCaracteres, temp.intLongueur, charContenu,
strlen(charContenu));
    return(temp);
}

void chaine::operator =(chaine &C) // l'objet courant à gauche du signe =
l'objet en paramètre à droite du =
{
    //strcpy(this->lesCaracteres,C.lesCaracteres); deprecated now
    strncpy_s(this->lesCaracteres,this->intLongueur, C.lesCaracteres,
C.intLongueur);
}

void chaine::operator =(char * charContenu) // l'objet courant à gauche du
signe = l'objet en paramètre à droite du =
{
    //strcpy(this->lesCaracteres,C.lesCaracteres); deprecated now
    strncpy_s(this->lesCaracteres,this->intLongueur, charContenu,
(int)strlen(charContenu));
}

void chaine::operator <<(char *charContenu)
{
    //strncpy(this->lesCaracteres, charContenu, strlen(charContenu));
deprecated now !
    strncpy_s(this->lesCaracteres, this->intLongueur, charContenu,
strlen(charContenu));
    //strncat(this->lesCaracteres,"\0",1); deprecated now !
    strncat_s(this->lesCaracteres,this->intLongueur,"\0",1);
}

int chaine::longueur()

```

```

{
    return ((int)strlen(this->lesCaracteres));
}
ostream & operator << (ostream & out, chaine &C)
{
    for (int i = 0; i < (C.longueur());i++)
    {
        out << C[i] ;
    }
    return out ;
}

chaine::~chaine()
{
}

#endif _CHAINE_CPP

```

```

// chaine.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "chaine.h"

int _tmain(int argc, _TCHAR* argv[])
{
    chaine maChaine(20);           // création de la chaîne maChaine de
longueur 20 caractères
    maChaine << " Hello World! ";   // place le mot "Bonjour" dans la
chaîne maChaine
    cout<<maChaine<<"\n";          // affichage du contenu de la chaîne
maChaine
    maChaine = "\0";               // efface le contenu de la chaîne
maChaine une autre variante possible pour l'affectation
    maChaine = " Hi, hi, hi !";    // place le message dans la chaîne
maChaine
    cout<<maChaine<<"\n";          // affichage du contenu de la chaîne
maChaine
    cout<<maChaine[1]<<"\n";        // affichage du contenu de la
position 1 de la chaîne maChaine
    cout<<maChaine.longueur()<<"\n"; // affichage de la longueur de la
chaîne maChaine

    chaine autreChaine(50);        // création de la chaîne autreChaine de
longueur 50 caractères
    autreChaine = maChaine + maChaine; // concaténation de deux chaînes
et affectation du résultat dans autreChaine
    cout<<autreChaine<<"\n";        // affichage du contenu de la
chaîne autreChaine
    chaine uneChaine(80);          // création la chaîne uneChaine longueur
80 caractères
    uneChaine = autreChaine + " un ensemble de caracteres!"; //
concatenation d'une chaîne et d'un ensemble de caractères
    cout<<uneChaine<<"\n";          // affichage du contenu de la chaîne
uneChaine
    return 0;
}

```