# M1 INF - C++

David Vanderhaeghe, Mathias Paulin, Hugues Cassé

2016-2017

# Chapitre 1

# TD1

## 1.1 Introduction

L'objectif de cette série de TD est d'approfondir les concepts vus en cours et préparer les TPs. L'accent est mis sur la programmation en C++, en exploitant tout le pouvoir expressif du C++, les subtilités de syntaxe et les possibilités du langage.

Chaque TD est composé d'une série d'exercices, qui sera débutée à une séance de TD, et à terminer pour la séance suivante.

Les exercices sont majoritairement tirés, traduits, adaptés du livre The C++ Programming Language, $4^{th}$ edition, Bjarne Stroustrup.

## 1.2 Création de classe

**Exercice 1.**

Définir une classe complex répondant aux spécifications suivantes :

- Cette classe représente un nombre complexe par un couple (partie réelle, partie imaginaire).
- Une instance de cette classe sera soit initialisée à 0 + 0i, soit à a + bi avec a et b des paramètres utilisateurs.
- Une instance de la classe devra pouvoir être copiée dans une autre mais aussi utilisée comme rvalue.

**Exercice 2.**

Donner le profil des opérateurs et méthodes suivants $+$, $-$, $*$, $/$, $\tilde{}$, real(), imag().

**Exercice 3.**

Implémenter les différents opérateurs et méthodes de la question précédente ~x est le conjugé de x et / est définir par x/z = (x*~z)/(z*~z)

**Exercice 4.**

On voudrait pouvoir écrire x.real() = 3 qu'est ce que cela implique sur votre opérateur complex::real().

**Exercice 5.**

On voudrait gérer les éventuelles erreurs. Identifier quels opérateurs peuvent produire des erreurs. Modifier la signature, et implémenter la gestion d'exception.

**Exercice 6.**

Définir les constantes complexes zero, i, one

**Exercice 7.**

Tracer les appels (à la main) lors des évaluations suivantes

```
const complex x {1.f, 2.f};
const complex y {2.f, 1.f};
complex z = x+y;

complex w = z/complex::one;
```

**Exercice 8.**

Proposez des fonctions surchargeant les opérateurs << et >> afin d'afficher et de lire des complexes.

**Exercice 9.**

Modifier la classe complex pour utiliser le mécanisme de **constexpr** et de référence à une r-value pour réduire les constructions inutiles.

## 1.3 Composition de classe

Nous allons créer un vecteur d'objets complexes, c'est-à-dire un conteneur linéaire et extensible. Pour ce faire, nous allons allouer un tableau d'une certaine taille $C$ dont nous n'utiliserons qu'une partie, $N$ (intialisé à 0 au démarrage).

**Exercice 10.**

Créez la classe CVector qui possède un constructeur auquel on peut passer la taille initiale du tableau $C$ en paramètre. Par défaut, la taille du tableau est 10. N'oubliez de créer le destructeur correspondant.

**Exercice 11.**

Les éléments du vecteur sont accessibles à partir de leur indice. On décide de surcharger l'opérateur [] qui prend en paramètre l'indice de l'élément accédé. Proposez une ou plusieurs version d'**operator**[ ] permettant de consulter seulement ou de modifier un élément du tableau. Doit-on déclarer ces fonctions **inline** ? Pourquoi ?

**Exercice 12.**

Avec la surcharge de l'opérateur **operator**[], on peut modifier le tableau mais il n'est pas possible d'ajouter de nouvel élément. On va maintenant écrire une fonction push_back() qui ajoute un nombre complexe à la fin du tableau. Donnez le profil de cette fonction et programmez la !

**Exercice 13.**

Surchargez l'opérateur << afin de pouvoir afficher le contenu d'un vecteur de complexe. Un problème va se poser, lequel ? Comment le corrigez-vous ?

**Exercice 14.**

Notre vecteur peut-être parcouru en utilisant des indices mais (a) ce n'est pas très standard, (b) cela demande un effort inutile au programmeur. Déclarez et définissez une classe interne iterator à CVector qui surcharge au moins les opérateurs du code suivant et ajoutez à CVector les classes nécessaires :

```
CVector v;
 ...
for(autor i = v.begin(); i != v.end(); ++i)
    cout << *i << endl;
```

**Exercice 15.**

Écrivez la méthode erase qui prend un iterator en paramètre et supprime l'élément pointé par l'itérateur du vecteur. Quels problèmes peuvent arriver ? Comment les réglez-vous ?

# Chapitre 2

# Héritage

**Exercice 16.**

Soit le programme suivant :

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

class B1 {
public:
    B1() { cout << "B1 constructed!" << endl; }
    ~B1() { cout << "B1 destructed!" << endl; }
};

class B2 {
public:
    B2() { cout << "B2 constructed!" << endl; }
    ~B2() { cout << "B2 destructed!" << endl; }
};

class D: public B1, public B2 {
public:
    D() { cout << "D constructed!" << endl; }
    ~D() { cout << "D destructed!" << endl; }
};

int main() {
   D d;
   return 0;
}
```

1. Qu'affiche l'exécution du programme ci-dessus ?
2. Si on utilise le main ci-dessus, qu'affiche-t-on maintenant ? Est-ce souhaitable ? Si non, comment peut-on corriger ?

```
int main() {
   B1 *b1 = new D();
   delete b1;
```

```
    return 0;
}
```

**Exercice 17.**

Soit le programme suivant :

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

class P {
public:
    void f1()   { cout << "f1 from P"; }
    virtual void f2()   { cout << "f2 from P"; }
    virtual void f3()   { cout << "f3 from P"; }
    void f1(int x) { cout << "f4 from P"; }
};

class Q : public P {
public:
    void f1() { cout << "f1 from Q"; }
    virtual void f2()   { cout << "f2 from Q"; }
};

class R: public Q { };

int main(void)
{
  R r;
  cout << "R:\n";
  r.f1();
  r.f2();
  r.f3();

  P *p = &r;
  cout << "P:\n";
  p->f1();
  p->f2();
  p->f3();

  return 0;
}
```

1. Qu'affiche l'exécution du programme ci-dessus ?
2. Toujours dans le programme ci-dessus, que se passerait-il si on ajoutait la commande `r.f1(0)` ?
3. Toujours dans le programme ci-dessus, que se passerait-il si on ajoutait la commande `P& pr = r;` ?
4. Toujours dans le programme ci-dessus, que se passerait-il si on ajoutait la commande `R& rr = pr;` ? Que faudrait-il écrire pour que ça marche ?

**Exercice 18.**

Soit le programme suivant :

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

class B {
public:
    virtual void show() = 0;
};

class D1: public B {
};

class D2: public B {
    void show() { cout << "ok!"; }
};

class D3: public D2 {
public:
    void show() { cout << "D2 says: "; show(); }
};

int main(void) {
    ...
    return 0;
}
```

1. Que se passera-t-il si on ajoute la commande B *b = **new** B ?
2. Que se passera-t-il si on ajoute la commande D1 *d1 = **new** D1 ?
3. Que se passera-t-il si on ajoute la commande D2 *d2 = **new** D2 ?
4. Que se passera-t-il si on ajoute les commandes B *p = **new** D2; p->show(); ?
5. Que se passera-t-il si on ajoute la commande D3 d3; d3.show(); ? Corrigez le programme pour que ça marche.

**Exercice 19.**

Nous désirons implémenter par une hiérarchie de classe des expressions configurables et évaluables. Notre objectif est de réaliser un logiciel où il est possible de saisir une fonction $f(x)$ et d'afficher la courbe correspondante sur un intervalle $[l, u]$ avec un pas $\epsilon$. Nous proposons de réaliser une hiérarchie de classes avec Expr comme classe de base, Var pour représenter la variable $x$, Const pour représenter une constante et Neg, Add, Sub, Mul, Div, Pow pour représenter diverses opérations.

```
class Expr {
    virtual float eval(float x);
    void print(ostream& out);
};

class Const: public Expr {
```

```
public:
    Const(float c): _c(c) { }
    virtual float eval(float x) { return _c; }
    virtual void print(ostream& out) { out << _c; }
private:
    float _c;
};
```

1. Que manque-t-il dans la déclaration de Expr ci-dessus pour qu'elle soit une classe de base acceptable ?
2. La déclaration des fonctions eval et print est incomplète, corrigez-les !
3. Définissez la classe var.
4. Définissez la classe Neg qui inverse la signe de l'expression qu'elle prend en paramètre. Un choix de gestion de la mémoire doit être fait : que décidez-vous ?
5. Avant de définir les classes d'opérateur binaire, définissez une classe BinOp qui prend en commun toutes les fonctionnalités communes aux classes d'opérateur.
6. Définissez les classes Add, Sub, Mul, Div et Pow.
7. Utilisez les classes dérivés de Expr pour construire la fonction $f(x) = x^2 - 3x + x/1$ qui sera stocké dans la variable $f$.
8. Dans le programme principal, demandez à l'utilisateur des valeurs pour $l$, $u$ et $\epsilon$ et affichez (a) la fonction $f$ et (b) les valeurs de la fonction $f$ sur l'intervalle $[l, u]$.

# Chapitre 3

# La suite...

### 3.0.1 Entrée/sortie

**Exercice 20.**
Ecrire un programme qui lit succéssivement un entier, un réel, un caractère et une chaine de caractères entrées au clavier. Ensuite votre programme affichera les valeurs des entrées lues au clavier, une sur chaque ligne

### 3.0.2 Conteneur linéaire standard

La bibliotèque standard du c++ offre un certain nombre de conteneur générique. Il sont générique dans le sens ou c'est au moment de la déclaration du conteneur que le programmeur spécifie le type des données qui sont contenu (par un mécanisme de template).

Les conteneurs classiquement utilisées sont :

— **std**::**vector**
— **std**:: **list**
— **std**::**deque**
— **std**::**map**

HOME WORK, allez voir les différentes page de manuel de ces structures, les compléxités des opérations usuelle (insertion, suppréssion, recherche).

Dans la suite de ce TD nous utiliserons uniquement **std**::**vector**, qui est un tableau dynamique, avec insertion en fin.

les opérations :

```
push_back();
pop_back();
resize();
emplace_back();
reserve();
at()
operator[](int)
```

pour parcourir une structure, il nous faut utiliser un itérateur (c'est le plus efficace pour tous les conteneurs, sans devoir se poser la question de comment il est implémenté).

```
for(std::vector<int>::iterator itr = myvec.begin();itr != myvec.end(); itr++){
        // do stuff !!
}
```

```
for(std::vector<int>::iterator itr = myvec.begin(), stop = myvec.end();
      itr != stop;we
      ++itr){
         // do stuff !!
}
```

**Exercice 21.**
Ecrire une fonction qui prend un vecteur en paramètre et insert n valeurs entrée au clavier dans ce vecteur.

### 3.0.3 auto

```
for(auto itr = myvector.begin(), stop = myvector.end();
     itr != stop;
     ++itr){
         // do stuff !!
}
```

L'itérateur "pointe sur" une valeur du tableau (ou la sentiennelle **std**::**vector**::end()). Pour accéder à cette valeur, vous pouvez passer par l'opérateur unaire $*$.

Par exemple $*$itr = 2.

Si vous utilisez l'intérateur uniquement en lecture, vous devez déclarer un const_iterator et l'initialiser avec cbegin()

**Exercice 22.**
Ecrire une fonction qui prend un vecteur en paramètre, et qui affiche son contenue.

**Exercice 23.**
Ecrire une fonction qui ajoute 10 à chaque element d'un vecteur d'entier

**Exercice 24.**
Ecrire une fonction qui ajoute une valeur passée en paramètre à un vecteur d'entier

**Exercice 25.**
Ecrire une fonction générique qui ajoute une valeur passée en paramètre à un vecteur

**Exercice 26.**
Utiliser cette fonction pour définir une fonction permettant d'ajouter un caractère en fin de chaine sur un vecteur de chaine de caractères (**std**::**string**) : appendChar(myStringVector, "a")

**Exercice 27.**
Proposez une solution en utilisant for_each

### 3.0.4 stl algorithm

### 3.0.5 Range-for

Depuis C++11, le langage permet l'utilisation de *range-for*. Il s'agit d'une écriture particulière d'une boucle **for**. Cette écriture permet d'exprimer simplement l'application d'opération à l'ensemble des éléments d'un conteneur.

```
for(auto val : myvector){
        // do stuff reading val !!
}

for(auto &val : myvector){
        //do stuff assigning val !!
}
```

**Exercice 28.**

Réecrivez la boucle **for** suivante avec une boucle **while** équivalente

```
// inputs are int max_length; char *input_line, int quest_count
int i;
for (i=0; i!=max_length; i++)
        if (input_line[i] == '?')
                quest_count++;
```

**Exercice 29.**

Réecrivez la boucle **for** pour utiliser un pointeur comme variable de contrôle, de tel sorte que le test soit de la forme *p=='?'.

**Exercice 30.**

Récrivez cette boucle pour utiliser un *range-for*

### 3.0.6 type utilisateur, operateurs

Implement an enum called Season with enumerators spring, summer, autumn, and winter. Define operators ++ and −− for Season. Define input (>>) and output (<<) operations for Season, providing string values. Provide a way to control the mapping between Season values and their string representations. For example, I might want the strings to reflect the Danish names for the seasons.

## 3.1 Exercices avancés

### 3.1.1 Passage de paramètre, reference, const

**Exercice 31.**

Read a sequence of possibly whitespace-separated (name,value) pairs, where the name is a single whitespace-separated word and the value is an integer or a floating-point value. Compute and print the sum and mean for each name and the sum and mean for all names.

**Exercice 32.**

Modify the program from to also compute the median.

**Exercice 33.**

Copy all even non-zero elements of an int[] into a vector<int>. Use a pointer and ++ for the traversal.

Write a program that strips comments out of a C++ program. That is, read from cin, remove both // comments and /* */ comments, and write the result to cout. Do not worry about making the layout of the output look nice (that would be another, and much harder, exercise). Do not worry about incorrect programs. Beware of //, /*, and */ in comments, strings, and character constants.

### 3.1.2 Gestion mémoire

new, delete
Allocation automatique (décoration de pointeur)

### 3.1.3 Analyse de code

```
void cpy_1(char* p, const char* q){
        int length = strlen(q);
        for (int i = 0; i<=length; i++)
                p[i] = q[i];
}

void cpy_2(char* p, const char* q)
{
        while (*p++ = *q++) ;
}
```