Calcul Scientifique et C++ : chausses trappes et peaux de bananes.

Stéphane Labbé

Université Joseph Fourier, Grenoble, Laboratoire Jean Kuntzmann.

Cours ANGD, le 30 septembre 2008.

Plan

En quelques mots...

Qui dit C++ dit bien entendu programmation orientée objets.

Pour le calcul scientifique, quels sont les avantages ?

- Grande souplesse de maintenance,
- souplesse de développement,
- gestion fine et modulaire des exceptions.

Quel en sont les désavantages ?

- Une conception plus complexe,
- une rigueur de documentation plus forte,
- des pièges pouvant détériorer les performances de calcul.

Ainsi, la première question à se poser est : Ai-je besoin de faire du C++ ?

Comparaison

Caractérisation de la programmation procédurale

- Non protection des données.
- Séparation des données et des fonctions.
- Rigidité des fonctions par rapport aux données.
- Répétition des algorithmes suivant le type de données.
- Objectifs linéaires de programmation.

Principes de base de la programmation orientée objets

- Présence de classes : définition de données pouvant interagir avec "l'extérieur".
- Encapsulation de données : protection des données, autonomie.
- Héritage : capacité des classes d'objets à se spécialiser à partir d'un modèle plus général.
- Polymorphisme : capacité des objets à réagir aux autres objets en fonction de leur classe.

Que choisir ?

Dans l'absolu, tout programme devrait être conçu en pensant objets.

Bien entendu, ce type de conception prend du temps, avant de se lancer dans l'aventure, les questions sont donc les suivantes :

- le projet doit-il être exploité par plusieurs personnes ?
- Est-ce une maquette pour une méthode ou un projet de développement à longue échéance ?
- Quel grain de modularité doit-on espérer ?

Diagrammes U.M.L., un exemple

L'outil premier de conception est le diagramme `U.M.L` : Unified Modeling Language.

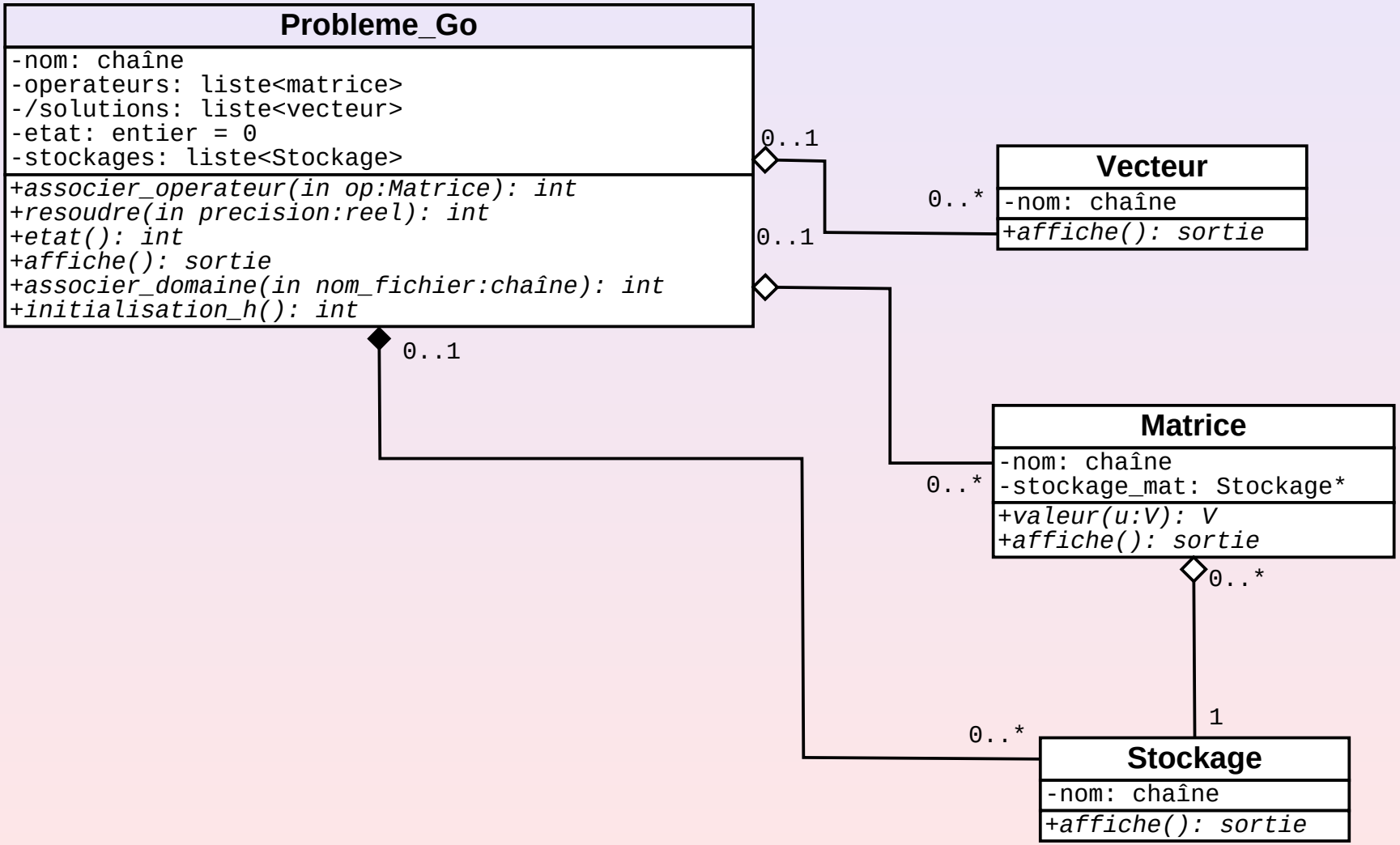
C'est un méta langage qui peut être utilisé efficacement pour la programmation orientée objets.

Les principaux atouts

- une conception graphique claire permettant des échanges simples entre participants à un projet,
- des outils de programmation pouvant exploiter les diagrammes (ex. : Dia, ROSE, Umbrella...),
- une indépendance au langage permettant une conception pérenne.

Bibliographie sur U.M.L. et son utilisation
P-AMuller et N. Gaertner(2000) Modélisation objet avec UML. Eyrolles.
M. Lai (2000). UML la notation unifiée de modélisation objet. Dunod Informatiques.
Un exemple de cours sur le web : http ://uml.free.fr
Le site officiel : www.uml.org
UML et calcul scientifique : le projet csimoon, http ://www-ljk.imag.fr/membres/Stephane.Labbe/Documents/csimoon.ps.gz

Diagrammes U.M.L.



A quoi est adapté le C++ ?

Création d'une structure modulaire de code

- avoir à disposition une structure facilement identifiable et réutilisable,
- pouvoir exploiter les sections de codes existantes et validées sans les retoucher,
- mettre sur pied des procédures de test des fonctions indépendantes.

Gestion simple des évolutions

- ajouter des fonctionnalités sans détruire l'existant, voire sans le modifier,
- pouvoir facilement intégrer des modules développés à l'extérieur, par d'autre personnes,
- rendre les logiciels réactifs à leur environnement.

Bibliographie sur U.M.L. et son utilisation

B. Stroutrup, Le langage C++, Pearson Education,
N. M. Josuttis, Object-Oriented Programming in C++, Willey.

A quoi n'est-il pas adapté ?

Les racines du calcul

Le C++, jusqu'à nouvel ordre, n'est pas réellement adapté au calcul sur de grandes structures linéaires.

A priori, un binaire donné pourrait être obtenu avec n'importe quel langage.

Ce n'est bien entendu qu'une vue de l'esprit car chaque compilateur a ses spécificités et plus un compilateur est généraliste, moins il risque d'être efficace pour une action ciblée : par exemple le calcul vectorisé sur des tableau qui est l'atout principal des compilateurs `FORTRAN`.

Une solution : l'interfacer les langages.

Les bibliothèques

Interfacer le FORTRAN et le C++ comporte de nombreux avantages :

- pouvoir bénéficier de bibliothèques FORTRAN de calcul scientifique particulièrement riches et fiables,
- réutiliser vos "morceaux" de codes programmés en FORTRAN tout en bénéficiant de la structure C++ de gestion des flux et des objets,
- capter le meilleur des deux compilateurs.

Non portabilité

Malheureusement, interfacer les langages dépend trop souvent des machines.

Une solution possible est alors l'utilisation des possibilités offertes par le C++ : les templates et les patterns que nous allons voir plus loin.

Principe : Utilisation d'objets comme drivers (voir par exemple le projet CSIMOON).

Opérateurs et mémoire

Une des possibilités emblématique du C++ : la surcharge d'opérateurs.

Cette idée de surcharger l'addition des matrices, par exemple, est particulièrement séduisante, mais entraîne des copies "parasites" qui peuvent être désastreuses dans un code de calcul.

Par exemple, pour l'exemple suivant

```
int main()
{
Complexe c1(1,2,"c1"),c2(2,3,"c2"),c3(4,5,"c3"),c4(0,0,"c4") ;
c4 = c1 + ( c2 + c3 ) ;
return 0 ;
}
```

Opérateurs et mémoire : résultat

On obtient !

```
Constructeur double/double/string pour l'objet c1.
Constructeur double/double/string pour l'objet c2.
Constructeur double/double/string pour l'objet c3.
Constructeur double/double/string pour l'objet c4.
Constructeur par copie créant l'objet copie de c3.
Constructeur par copie créant l'objet copie de c2.
Constructeur double/double/string pour l'objet copie de c2 + copie de c3.
Constructeur par copie créant l'objet copie de c1.
Constructeur double/double/string pour l'objet copie de c1 + copie de c2 + copie de c3.
 Destruction de l'objet copie de c1 + copie de c2 + copie de c3.
Destruction de l'objet copie de c1.
Destruction de l'objet copie de c2 + copie de c3.
Destruction de l'objet copie de c2.
Destruction de l'objet copie de c3.
Destruction de l'objet copie de c1 + copie de c2 + copie de c3.
Destruction de l'objet c3.
Destruction de l'objet c2.
Destruction de l'objet c1.
```

Méta-programmation

Dans cet exemple, une classe totalement template est créée.
Il n'y a pas de .C !.

Quand le nombre de composantes est trop grand, le compilateur rend l'âme.
Ce type de pratique n'est en fait adaptée que pour les petits vecteurs. !

Voir TP03-05

L'utilisation des patrons de classes.

La S.T.L. : La librairie standard

On y trouve des algorithmes agissant sur des conteneurs. Un conteneur est une collection d'objets.

Aux conteneurs sont associés des itérateurs qui permettent de les parcourir.
Quelques exemples de conteneurs (voir TP03-03.1-4)

- vector
- list
- set
- multiset
- map
- multimap

Bibliographie, un exemple

N. M. Josuttis, The C++ standard Library : a tutorial and reference, Addison Wesley.

valarray

Un conteneur mathématique : `valarray`

C'est le conteneur dont le comportement a été conçu pour être apparenté à celui des "vecteurs numériques".

Fonctions de spécification des sous ensembles :

- `slice_array`
- `gslice_array`
- `mask_array`
- `indirect_array`

Ce sont des classes auxiliaires qui sont créées par la classe `valarray` mais ne peuvent pas être utilisées directement dans les sources d'un code de calcul.

valarray

Quelques exemples de déclarations

```
valarray<int> v0 ;
valarray<double> v1(100) ;
valarray<double> v2(1.2,1000) ;
valarray<double> v3=v2 ;
const double donnee[]={1,2,3,4,5} ;
valarray<double> v4(donnee,3) ;
```

Le type `slice` permet, entre autres, d'extraire des données d'un `valarray` dans un format proche de celui utilisé pour `BLAS` : il donne des objets du type `slice_array`.

Attention : Un `slice_array` n'est qu'une référence sur une partie de tableau et en particulier ne pourra pas être copié.

Patterns, un exemple

Les *Patterns* sont des prototypes de classes permettant de définir des rôles de classes.

Par exemple : la classe singleton qui permet d'avoir une classe n'ayant qu'une instance, ou encore la classe factory permettant de générer des instances de classes à la volée.

Un exemple : le singleton

Permet de créer une classe rémanente (TP 3-04).

Bibliographie, un exemple

D. Vandervoorde, N. M. Josuttis, C++ templates : the complete guide, Addison Wesley.

Les messages d'erreurs et procédures d'exception

Le C++ possède une fonctionnalité de gestion des exceptions totalement modulable.

Dans cet exemple, une gestion de l'overflow est implantée.

```
for (i_monomes=–monomes.end() ; (i_monomes !=monomes.end())&&(i_monomes->degres()>=n) ; –i_monomes)
{
res=res*puiss(z,d-(i_monomes)->degres()+n)+i_monomes->coeff()*double(facto((i_monomes)->degres()))/double(facto((i_monomes)->degres()-n)) ;
throw OverFlow("*") ;
d=i_monomes->degres()-n ;
}
```

TP04-02

Les messages d'erreurs et procédures d'exception

Gestion des exceptions dans le programme principal :

```
int main()
{
try {
 ...
cout << P.valeur_deriv(1,199) << endl ;
...
}
 catch (OverFlow &O)
{
O.debug_print() ;
}
 cout << "C'est reparti." << endl ;
}
```