Eléments de C++/CLI

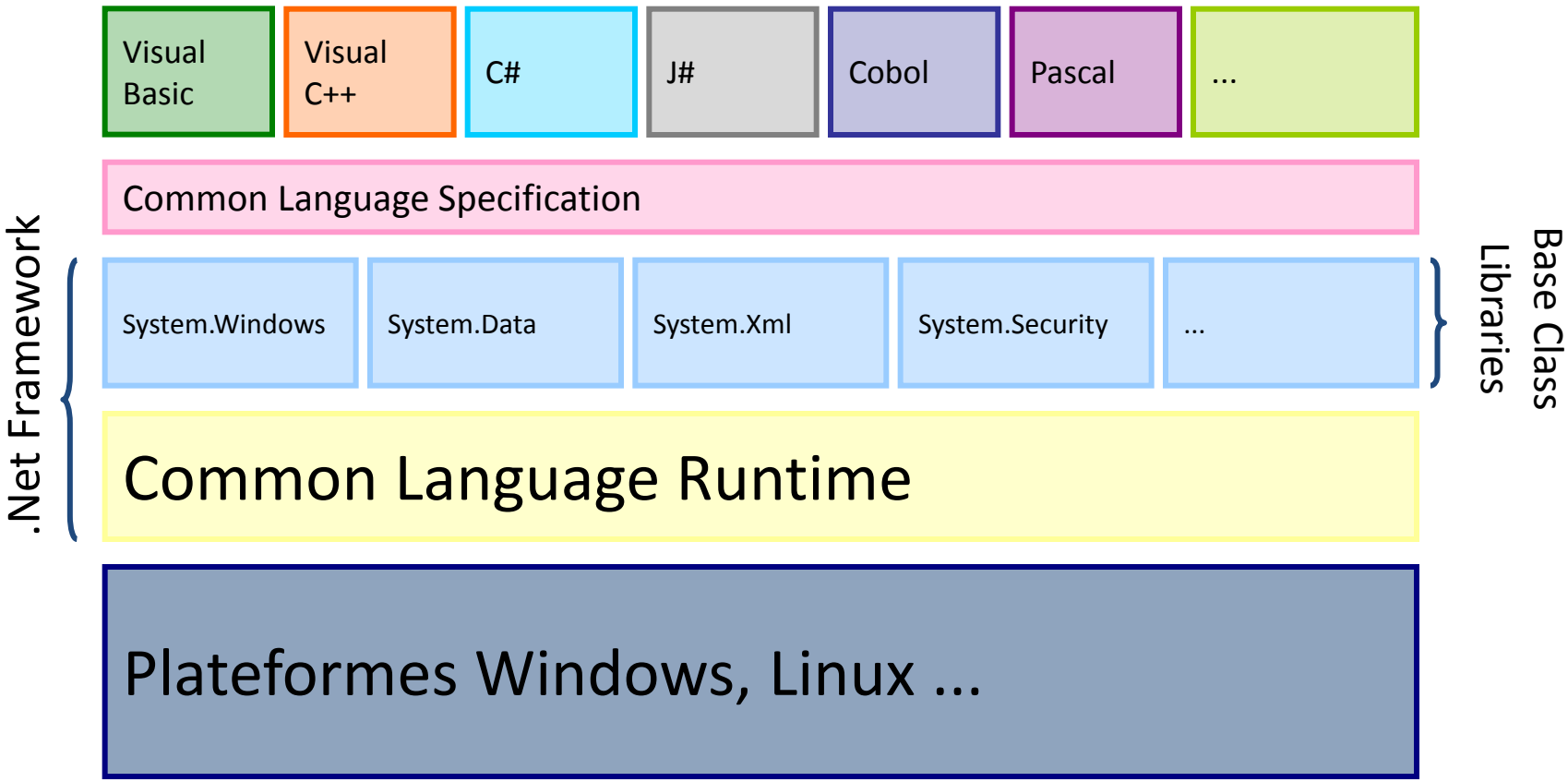
Présentation

Pour implémenter des projets Monte Carlo « user-friendly »

- Code implémenté avec la plateforme .NET:
  - Support pour les Windows application forms, ASP.NET
  - Base Class Library: très simple de récupérer, transformer, afficher des données
  - Programmes assez efficaces, mais moins que les programmes C++
- Morceaux de code pour les projets Monte Carlo
  - Uniquement en C++ natif

Comment faire interagir ces codes efficacement?

Quelques rappels



Common Language Runtime

S'occupe de tout:

- Compilation IL
- Vérification de types
- Exceptions
- Débuggeur
- Interopérabilité COM
- **Ramasse-miettes**

C++ standard (natif)

- Plus d'efficacité
- Très répandu
- Programmation « bas-niveau » possible
- Arithmétique sur pointeurs possible
- Mais l'utilisateur doit gérer le cycle de vie de ses objets.

Problème pour Microsoft: comment interfacer du code .NET et du C++ pour une transition C++ -> .NET en douceur?

Premières solutions

Platform Invoke (P/Invoke)

- Vu pendant le projet .NET
- Peut être employé pour d'autres langages (ex: du C)
- Mais:
  - Uniquement pour utiliser une dll native depuis du code managé
  - Performance dégradée
  - Ne permet que d'invoquer des fonctions « exportées »
  - Impossible de manipuler des classes définies dans la dll

Managed C++

- Première solution de Microsoft pour une vraie interface C++/.NET
- Exemple:

```
__gc class M
{
   __event ClickHandler* OnClick;
public:
   __property int get_Num()
   {
      return 0;
   }
   __property void set_Num(int)
   {
   }
};
```

- Flop

C++/CLI

- Nouvelle solution de Microsoft pour l'interface C++/.NET
- Langage de programmation de plus bas niveau de la plateforme .NET
- Sensé être le langage le plus puissant de la plateforme
- Politique de Microsoft: surtout pour l'interfaçage
- Permet de:
  - Compiler du code natif
  - Accéder à la BCL
  - Mélanger du code natif et managé

Exemple

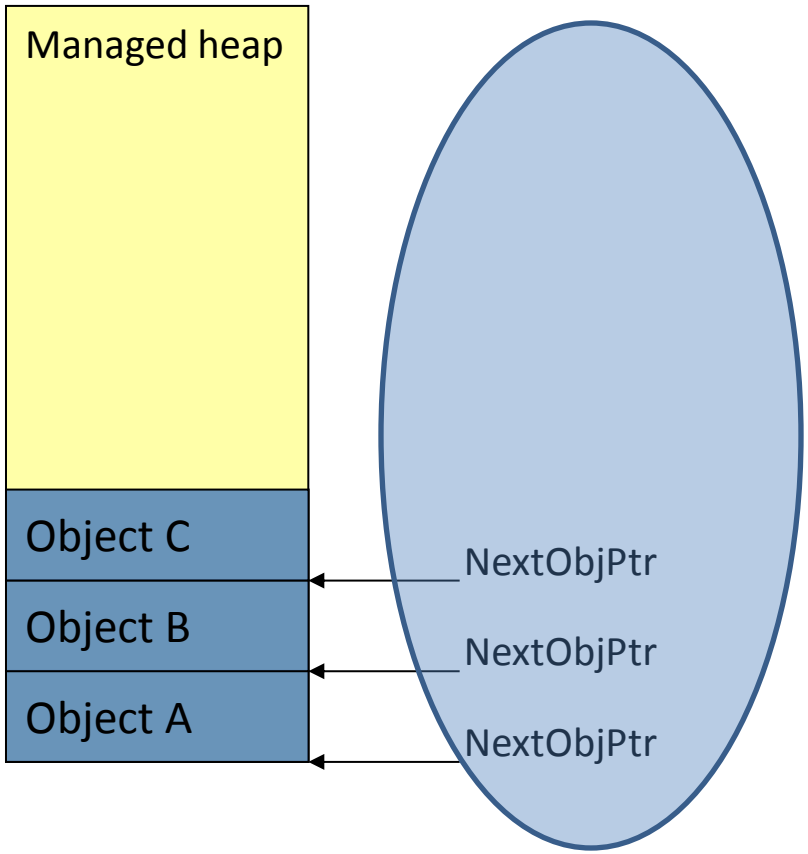
Mélanger du C, du C++ et du C++/CLI

```
#include <stdio.h>
#include <iostream>
int main()
{
	// fonction C pour imprimer "hello"
	printf("hello");
	// objet C++ pour imprimer une virgule suivie d'un espace
	std::cout << ", ";
	// objet .NET pour imprimer "world"
	System::Console::WriteLine("world");
}
```

Description du langage

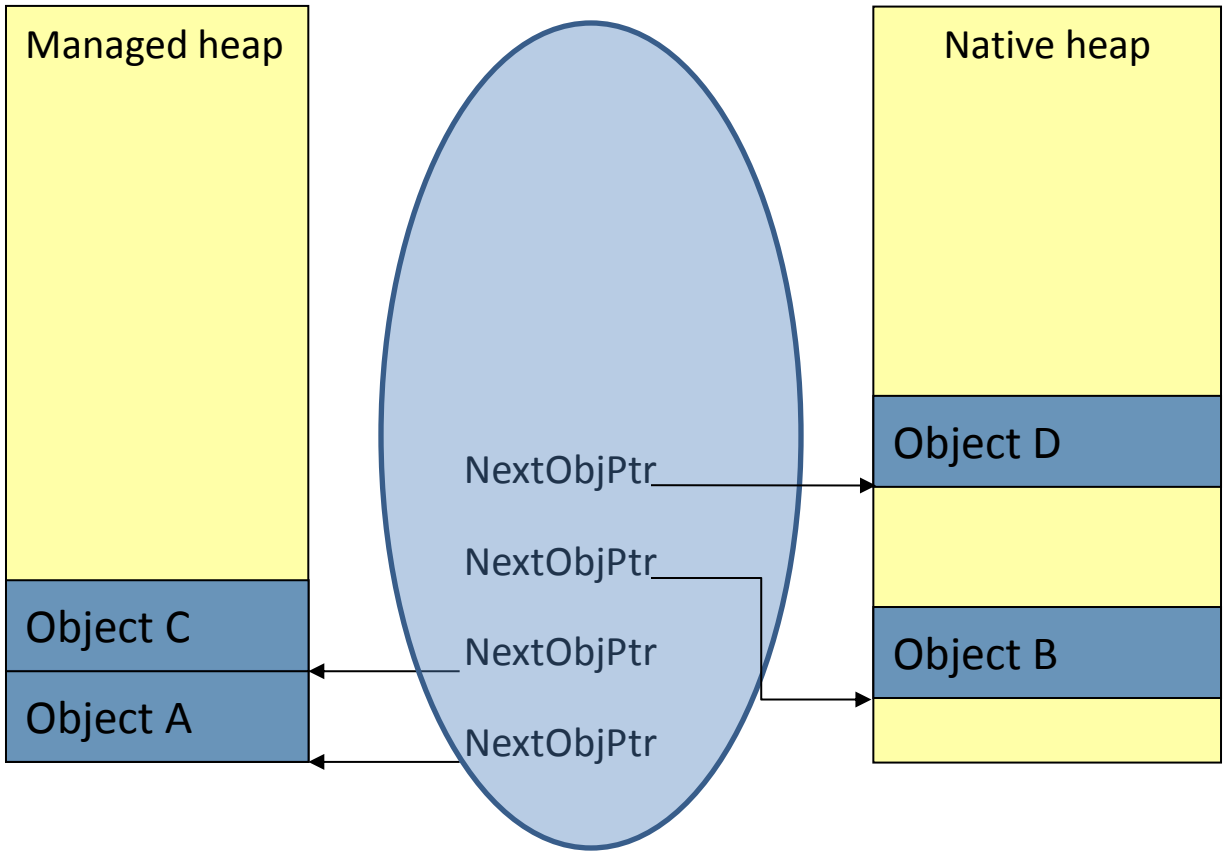
Pour comprendre la suite

Pour un langage .NET standard (ex: C#)



Pour comprendre la suite

Pour le langage C++/CLI



Différences

Tas non managé: pointeurs habituels sur les objets

- Pointent vers une zone mémoire
- Possible de faire de l'arithmétique dessus
- Si l'objet pointé est détruit, on ne sait pas vers quoi on pointe

Tas managé: références vers les objets

- **Ce ne sont pas des pointeurs!**
- On parle de « handle » vers un objet
- Peuvent faire référence à plusieurs emplacements mémoire au cours de l'exécution du programme

Non-managé/managé: mots-clés

Déclaration de classe native	Déclaration de classe managée

class ClasseNative{
ClasseNative();
~ClasseNative();
(…)
}
	ref class ClasseManagee{
	ClasseManagee();
	~ClasseManagee();
	(…)
	}

Pointeurs ClasseNative* clNat = new ClasseNative(); clNat->InvoqueMethode();	**Handles** ClasseManagee ^clMan = gcnew ClasseManagee(); clMan->InvoqueMethode();
Déréférencement, références… ClasseNative clNat2 = *clNat; ClasseNative& clNat3 = clNat2; ClasseNative* clNat4 = &clNat2;	**Tracking references** ClasseManagee clMan2 = *clMan; ClasseManagee% clMan3 = clMan2; ClasseManagee^ clMan4 = %clMan2;

Mélanger du C++ et du C++/CLI

On peut créer des classes managées et natives qui coexistent. Mais elles n'interagissent pas encore.

- Comment effectuer des manipulations de pointeurs avec des objets managés?
- Comment invoquer une fonction native avec des paramètres managés?
- Comment invoquer une fonction managée avec des paramètres natifs?

Pointeurs intérieurs

Exemple

```
ref class ClasseManagee{
        int valeur;
        public ClasseManagee() {};
};

int main(array<System::String ^> ^args)
{
        for (int i=0; i<100000; i++)
                gcnew ClasseManagee();
        ClasseManagee^ d = gcnew ClasseManagee();
        d->valeur = 100;
        interior_ptr<int> pint = &d->valeur;
        printf("%p %d\r\n",pint,*pint);
        for(int i=0; i<100000; i++)
                gcnew ClasseManagee();
        printf("%p %d\r\n",pint,*pint);
   return 0;
}
```

Output:

012CB4C8 100
012A13D0 100

Propriétés

Pointeurs natifs implicitement convertis

```
void methodeManagee(interior_ptr<int> param){   (…)  }

int main(array<System::String ^> ^args){
        ClasseManagee^ d = gcnew ClasseManagee();
        interior_ptr<int> pint = &d->valeur;
        methodeManagee(pint); // pointeur intérieur -> OK
        int x = 8;
        methodeManagee(&x); // pointeur natif -> OK
}
```

Arithmétique sur pointeurs intérieurs possible

```
int main(array<System::String ^> ^args){
        array<int>^ arr = gcnew array<int> {2,4,6,8,3,5,7};
        interior_ptr<int> p = &arr[0];
         int s = 0;
        while (p != &arr[0] + arr->Length)
        {
                s += *p;
                p++;
        }
        printf(« Somme= %d\r\n",s);
}
```

Pointeurs épingles

Exemple

```
int main(array<System::String ^> ^args)
{
        for (int i=0; i<100000; i++)
                gcnew ClasseManagee();
        ClasseManagee^ d1 = gcnew ClasseManagee();
        for (int i=0; i<100000; i++)
                gcnew ClasseManagee();
        ClasseManagee^ d2 = gcnew ClasseManagee();
        interior_ptr<int> intptr= &d1->valeur;
        pin_ptr<int> pinptr = &d2->valeur;
        printf("intptr = %p, pinptr = %p\r\n", intptr, pinptr);
        for (int i=0; i<100000; i++)
                gcnew ClasseManagee();
        printf("intptr = %p, pinptr = %p\r\n", intptr, pinptr);
   return 0;
}
```

Output:

intptr=012CB4C8, pinptr=012CE3B4
intptr=012A13D0, pinptr=012CE3B4

Propriétés

Implicitement convertis en pointeurs natifs

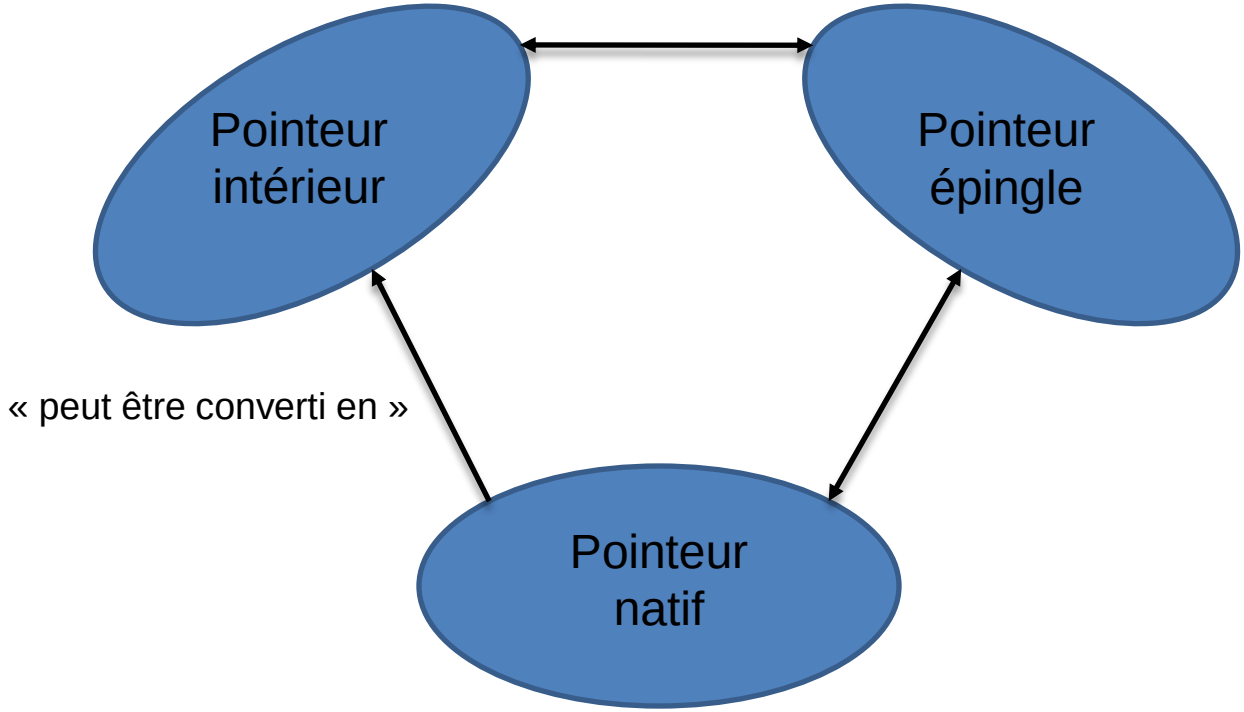
```
#pragma unmanaged
// Initialisation d'un tableau de taille 10 (code natif)
static void methodeNative(int* p) {
        for(int i = 0 ; i < 10 ; i++)
        p[i] = i;
}

#pragma managed
int main(array<System::String ^> ^args){
        array<int>^ arr = gcnew array<int>(10);
        pin_ptr<int> p = &arr[0];
        methodeNative(p); // le tableau arr est initialisé
}
```

Danger: fragmentation du tas managé

- Utiliser des pointeurs épingles uniquement quand c'est nécessaire

Résumé des conversions



Membre natif dans classe managée

```
class MembreNatif{…};
ref class MembreManage{…};

ref class ClasseManagee {
public:
    MembreManage mm;
    MembreManage ^hmm;
    MembreNatif mn; // Erreur: impossible d'avoir une instance de classe native comme membre
    MembreNatif *pmn; // OK: un pointeur non managé est accepté
    int tableauNatif[5] = {2,4,3,5,3}; // OK: tableaux natifs sont également acceptés
};
```

Handle déclaré dans classe native

```
ref class MembreManage{…};

class ClasseNative {
public:
        MembreManage ^hmm; // Erreur
};
```

Solution:

```
#include <msclr/auto_gcroot.h>

class ClasseNative {
public:
        auto_gcroot<MembreManage^> hmm;
};
```

Un mot sur les TP

Utiliser VS2010, pas VS2012!

- Le code:
  - Doit être commenté
  - Doit être structuré correctement
  - Test: on doit pouvoir exécuter le code après un copier/coller vers une autre machine
- **TP1: sur ensiwiki**
  - Prise en main des outils