Bonjour tout le monde

Objectif :

- Être connecté sur le poste PLB

=> Installez Visual Studio Community + C++ Dev Desktop en C++

https://visualstudio.microsoft.com/fr/thank-you-downloading-visual-studio/?sku=Community&rel=17&utm_medium=MSDN&utm_source=VS&utm_campaign=MSDN&utm_content=link

OU - Poste local de votre convenance -

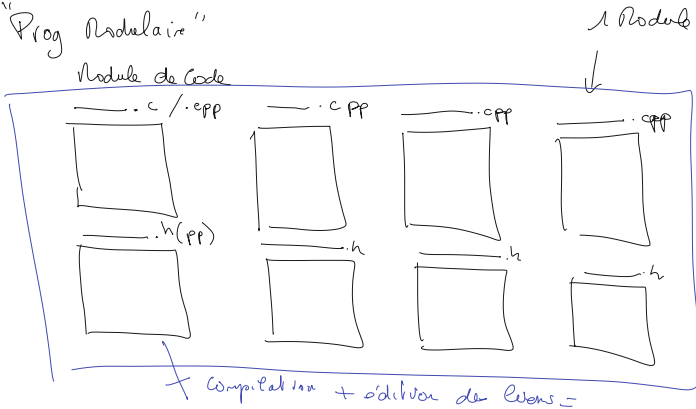
- avoir un Compilateur C++ moderne.

>14

- Dev from scratch
- Pseudo développement C++ avec un compilateur C
- (UML et diagramme de classes)
- Notation de lien objet / mémoire et type de mémoire
- Les design patterns du Gof
- CVE et SANS (les failles)
- Dév d'interfaces sous Windows (WIN32, MFC, Winform, C++ et C++/CLI)
- Les outils d'analyse statique de code (SonarQube, Valgrind)
- Les outils de détection dynamique
- Git
-

Bases

"Prog Modulaire"



Pb d'entrée → exécutable contrairement à une lib/ bibliothèque

↳ int main (int argc, char *** argv)

.c → mode de compilation → Langage C

.cpp → ↘ sinon cpp.

C → 70/80 → Méta Langage ↑ assembleur

assembleur → C → Code (presque) aussi performant

bcp + productif.

C est donc une solution Technique

→ qques bibliothèques. (libc / ANSI)

ANSI = Compilation sous ≠ OS.

ex memcpy
fopen → Windows
Unix
Linux
- MacOS (≤ 9)

- différence selon les environnements.
  - ANSI = Commun
  - Architecture = Intel x86/64 - AMD
    - ARM / QUALCOMM
    - PBCA - PowerPC (~~CISC~~ RISC)
    - etc....
- Little endian / big endian

Code → WORD

byte → 8 bits

(court) WORD (long) 16/32/64 bits
short int → long

e, z, 8x octet

0x ff00 => 65 280

ff 00 — Poids fort d'abord

00 ff — poids faible d'abord

Ecrire du code portable

Concept = simple

dans la pratique = complexe

C => Prog structurée

→ Modules

→ fonctions

→ variable - locales / parametres -
. globales
↙
mauvaise pratique
à bannir si possible ....

Code → bibliothèques fonctionnelles

ex: libc (ANSI)

libc → Pas de . solutions de dev
Sécurité
- implémentation spécifique pour "faire mieux"

C - (Prog structuré)

↘ C++ = C + classes / objets

→ Compilateur cpp, "c'est tout"

- est il possible de faire du pseudo objet en C ?

Bouh ! structure + pointeurs

- Concepts objet ⇒ années 80 (SmallTalk)

C → "pseudo objet"

⇓

devenir du C++.

Apports des concepts objet ?

Concepts, Abstraction, modélisation, encapsulation. Patterns. ...

Concepts objet

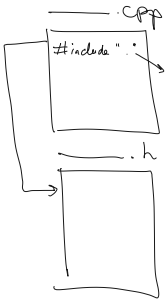
Modélisation : UML

9/12 diagrammes
- statique
- temporel

→ apprendre à écrire.
→ inviting à lire.

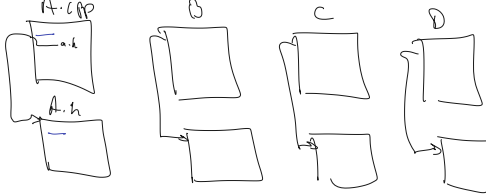
⇒ diagramme de classes.

(C) → - Compilateur C++
- fo
- pour d'entrée

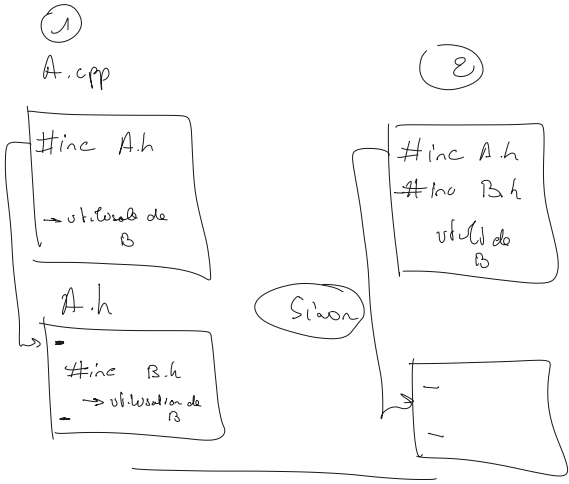


→ Implémentation
Valider le code implémenté / déclaré

→ déclaration
des interfaces
↳ classe, prototype
(fonctions > membres > méthodes)
→ code inline possible.

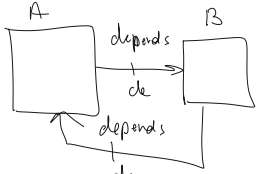


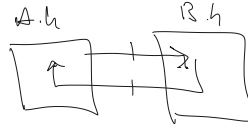
- l'inclusion de B.h dans le module A
  - si il y a des dépendances de B pour A.h → a inclure dans A.h
  - sinon, il est recommandé de l'inclure après le #include "a.h"
    → pour limiter les dépendances -



Quid si

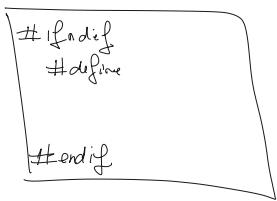
A.h a besoin de B.h } - Références
B.h a besoin de A.h - Circulaires





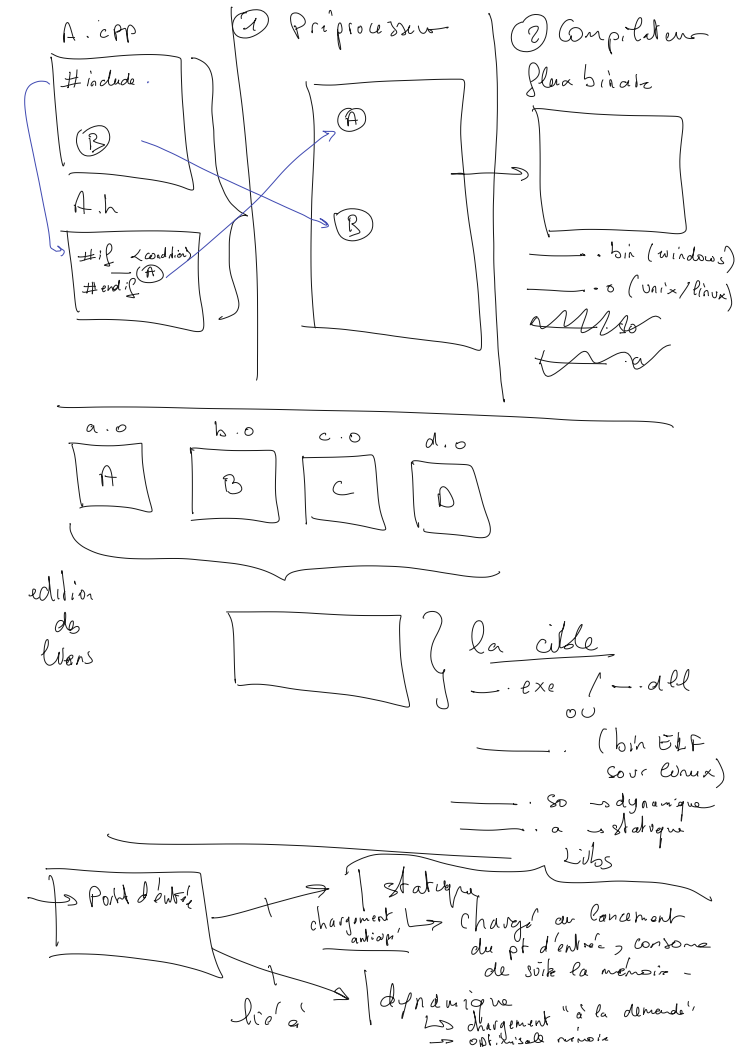
Solution

- PRAGMA ONCE (Pas standard)



C/C++ → Compilation à 2 passes -

- 1ère passe = "préprocesseur" → mise en forme du code et compilation instructions # qui sont traitées.
- 2nd passe = Compilation (d'un code non visible)
- Edition des liens - (en final)


A.cpp
#include
B
A.h
#if <condition>
A
#endif
1 Préprocesseur
A
B
2 Compilateur
flux binaire
. bin (windows)
. o (unix/linux)
. so
. a
a.o
b.o
c.o
d.o
A
B
C
D
edition des liens
la cible
. exe / . dll
ou
. (bin ELF sous Linux)
. so → dynamique
. a → statique
Libs
Point d'entrée
statique
chargement anticipé
→ Chargé au lancement du pt d'entrée, consomme de suite la mémoire
lié à
dynamique
→ chargement "à la demande"
→ optimisation mémoire

C++ de base = C + classes et objets

↓

"Solution technique génératrice de bugs"

C++ "moderne" (≥ C++ 11)

↳ l'utilisation des bibliothèques ANSI

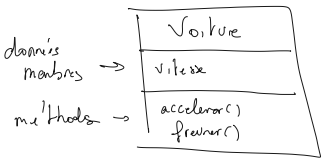
Pour les fonctionnalités est devenu OBSOLÈTE

- NON RECONNANDE = on peut m'avoir faire (patterns)
- DEPRECIÉ = une mise à jour sera à faire, ce n'est pas une erreur - Etat de l'art, pattern = OK
- OBSOLÈTE = Toujours disponible, mauvaise Pratique (anti pattern)
- ILLEGAL = (options) à activer/désactiver WARNING ! ERRORS -

- Écrire du "C++" en C

1 classe
↳ n instances
/ créées / détruites

UML



PascalCasing
camelCasing
lowercasing
UPPERCASING

snake_casing
kebab-casing

Problème / Besoin → Analyse → Patterns

↓

Définir

↓

Implémenter.

(C → Pbm → implémentation)

Notation UML. Unified Modeling Language.

9/12 Diagrams:

→ statique
- classe
- UseCase

→ Dynamique
- séquence

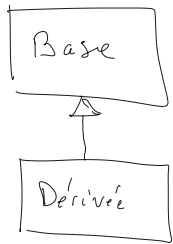
≪ données membres
- champs
- attributs
- propriétés

⇐ fonctions membres
- méthodes
- verbes

} membres

- Propriétés / accesseurs (getters setters) → Méthodes qui ressemblent à des champs.
- Événements / events → champ qui pointe vers une méthode.

- Encapsulation
  - "mettre dans une capsule"
  - "protéger" les membres de
    accès non prévus
    - private → uniquement le type en cours
    - protected → privé + types hérités
    - public → Tout ouvert
  - package / internal / file (Java / C#)
- Héritage (Généralisation)

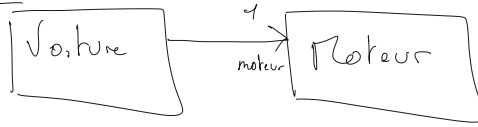


Verbe être
Dérivée "est un" Base
= Base + extensions / redéfinitions.

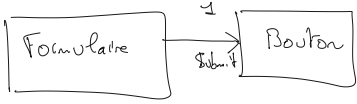
- Aggrégation

——— = ⟷ - bidirectionnel
⟵ unidirectionnel
→ nom du rôle / attribut, → cardinalité

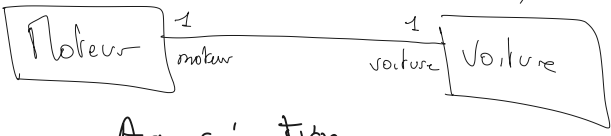
ex :



"navigation" : la Voiture voit 1 moteur de Type Moteur



" a un ... " (avoir)

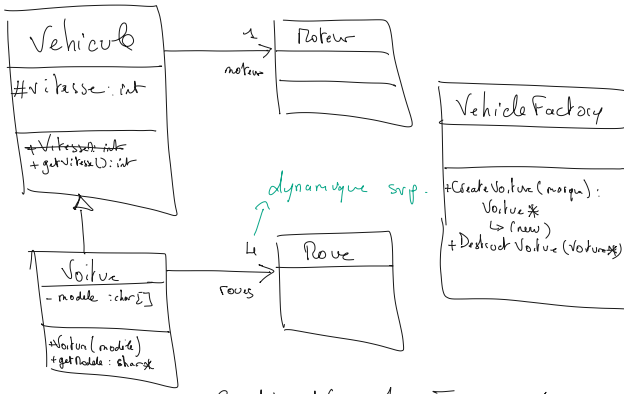


→ Aggrégation

"est ce que la Voiture crée un Moteur ?"
généralement NON → pas d'instanciation dans le conteneur - ?!!

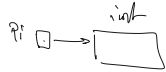
- il faut utiliser un objet au même niveau que là où il est créé

- Fabrique d'Objets = Factory
  → Recettes de bonnes pratiques
  = Design Patterns

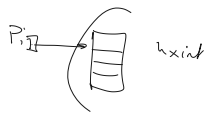


Cardinalité : 1 : Type scalaire
4 : Type Tableau
(n : Type Collection)

```
int *pi = new int;
delete pi;
```




```
int *pi = new [4] int;
delete [] pi;
```



```
#pragma once

class Roue
{};

class Moteur
{};

class Vehicle
{
public:
 int vitesse;
 Moteur* moteur;
public:
 int GetVitesse();
};

class Voiture : public Vehicle
{
public:
 Roue* roues;
};

class VehicleFactory
{
public:
 Voiture* CreateVoiture(char* modele);
 void DestroyVoiture(Voiture* voiture);
};


// ConsoleApplication2.cpp : Ce fichier contient la fonction 'main'. L'exécution du programme commence et se termine à cet endroit.
//

#include <iostream>

#include "ConsoleApplication2.h"

int Vehicle::GetVitesse()
{
    return vitesse;
}

Voiture* VehicleFactory::CreateVoiture(char* modele)
{
    Voiture *tmp = new Voiture();
    tmp->moteur = new Moteur();
    tmp->roues = new Roue[4];

    return tmp;
}

void VehicleFactory::DestroyVoiture(Voiture* voiture)
{
    // Démanteler :
    delete[] voiture->roues;
    delete voiture->moteur;
    delete voiture;
}

int main()
{
    std::cout << "Hello World!\n";
}
```

Aggrégation (faible)

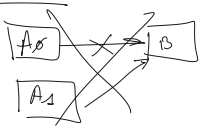
= pas de relation forte entre A et B

Aggregation forte / Composition

si: 

mais que:
- la durée de vie de A = celle de B
- que B n'est jamais détruit explicitement
- que B ne sera jamais transmis à une autre instance

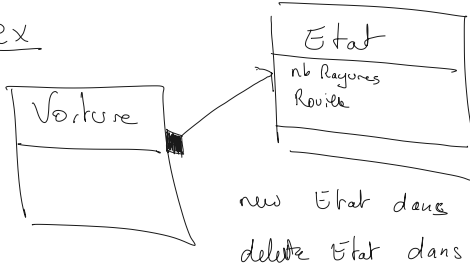
dvpg d'instances:

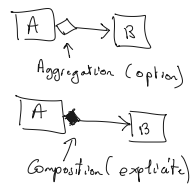


- Alors il est souhaitable d'instancier / détruire B dans A.

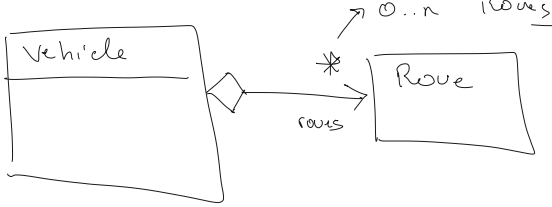
- Ne pas confondre une instance de Collection et le Contenu de cette Collection
- La Collection sera toujours une Composition

ex





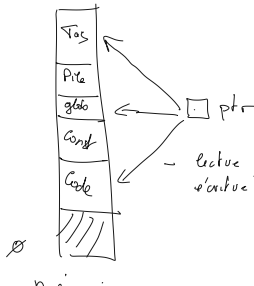
new Etat dans le ctor() de A
delete Etat dans le ~A



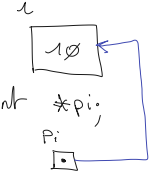
- Roues → Aggrégation
- Collection → (attribut)
  ↳ new et delete dans la Classe -

Types de mémoire

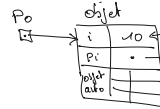
- Tas → Tout le reste
- Statiques → globales - attributs de classe -
- Pile → Taille fixe (définie à la compilation)
  - Type de gestion -
- Code → ni lecture / ni écriture -
- Constantes → lecture seule -
- Réservée → lecture / écriture → peut planter l'appli si accès -




```
int i = 10;
```




```
void *po = new Objet();
```




```
void *po = new Objet(); //recommandé
```

Objet o; ⇒ Objet automatique
↳ pas besoin de delete

```
int m(int i)
{ i++;
}
```

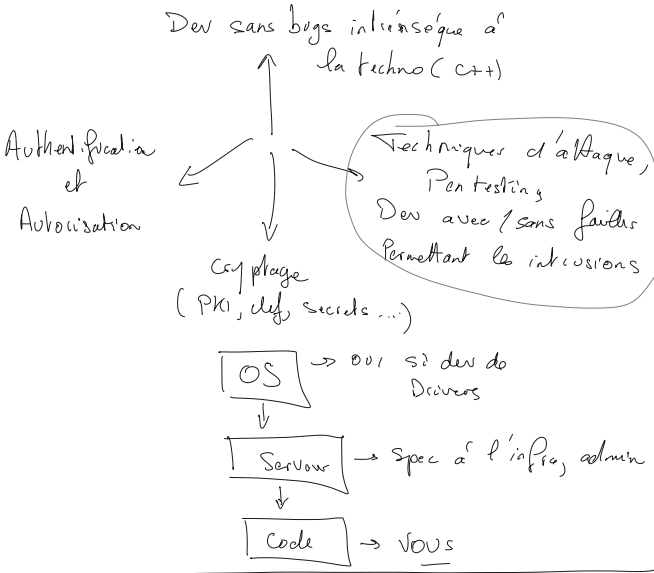
```
int n(int &i)
{ i++;
}
```

```
int i=10;
int *pi = &i;
*pi++
```

i=10 → m(i) → n(i) → i=11 ?

i = ?

La Sécurité

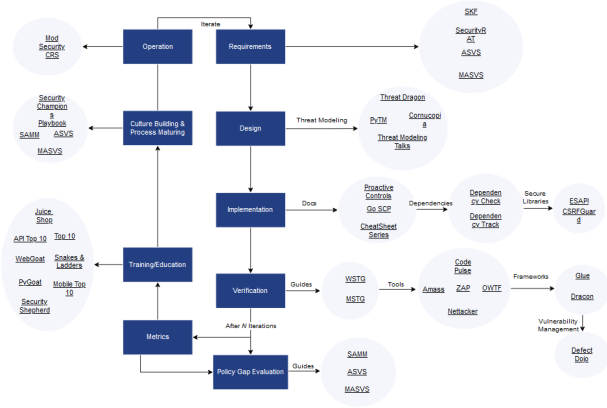


---

(1) Renforcement : https://www.cisecurity.org/cis-benchmarks/

(2) Bonnes pratiques de Dev :

La chaîne de la sécurité Web avec OWASP : https://owasp.org/projects/



Le référentiel Européen : https://www.enisa.europa.eu/

Pour vous : Les CWE : https://cwe.mitre.org/

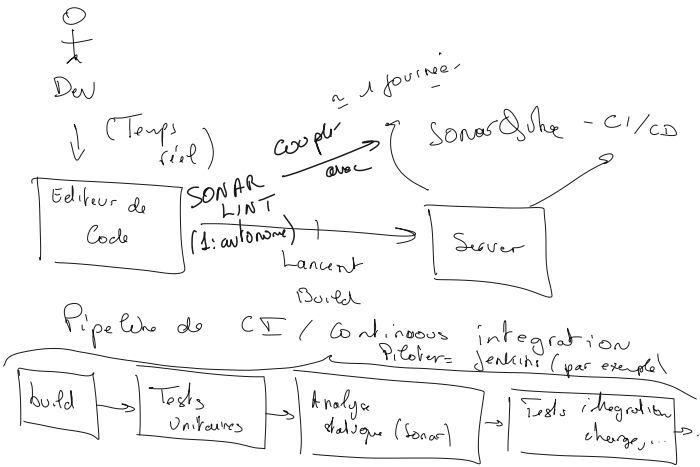
Le TOP25 : https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html

C++ Coding standard (pour s'en inspirer) : https://resources.sei.cmu.edu/downloads/secure-coding/assets/sei-cert-cpp-coding-standard-2016-v01.pdf

Guidelines pour du C++ moderne :
https://www.autosar.org/fileadmin/user_upload/standards/adaptive/17-03/AUTOSAR_RS_CPP14Guidelines.pdf

Faille de "Type" Composants avec des Vulnérabilités
→ Voir les CVE
→ Outils de scan et dépendances / SonarQube par ex.

Liste des CVEs : https://cve.mitre.org/index.html
La base NVD : https://nvd.nist.gov/ (dont CVEs)
Recherche : https://nvd.nist.gov/vuln/search

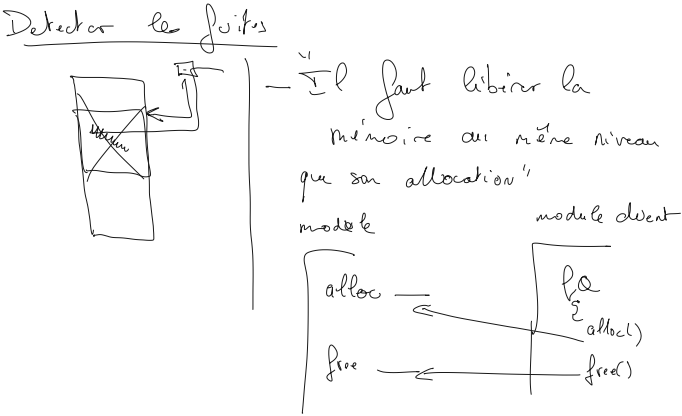


Règles Sonar :
https://rules.sonarsource.com/cpp/quickfix

Installation de Lint sous Visual Studio :
https://marketplace.visualstudio.com/items?itemName=SonarSource.SonarLintforVisualStudio2022
Environnements : VSCode, Visual Studio, Eclipse, IntelliJ

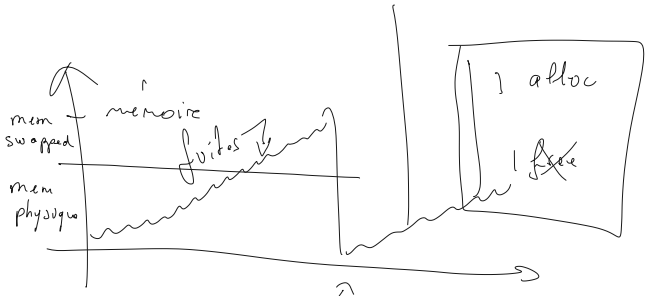
Outils d'analyse statique de process / code c++
deepSource, Perforce, SemGrep, Sonar, ...
Cppcheck

Outils d'analyse dynamique de code :
Valgrind
Visual Leak detector
Deleaker



foo ()
{
  alloc()
  .
  .
  ;
}

" a priori, il y a une fuite "



- Fuites mémoire
- Code Compatible multi-système
- Code Robuste
- Concepts trop complexes (thrichage d'allocateur, virtuelles, pures, Templates, gestion mem, Pointeurs, Refs, ...)

- Gestion de la Mémoire par le Dev

C/C++ en final

ne "respecte pas le cahier des charges"

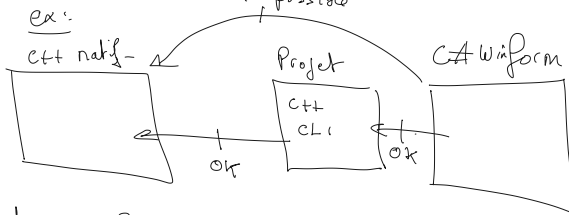
Mais.
- empreinte mémoire excellente
- excellentes performances

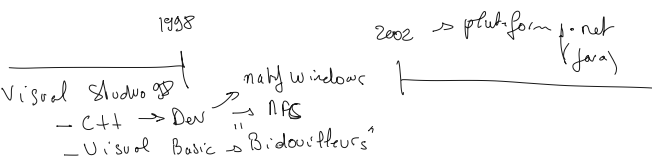
Solutions = Technos "gérées"
- Java
- C#
- Python .....

C++/CLI

Implémentation du language C++ qui génère du code géré .net CLR
→ pas de gestion de la mémoire
→ pas du gains propre au C++ natif

Permet l'interopérabilité entre le C++ et le .net



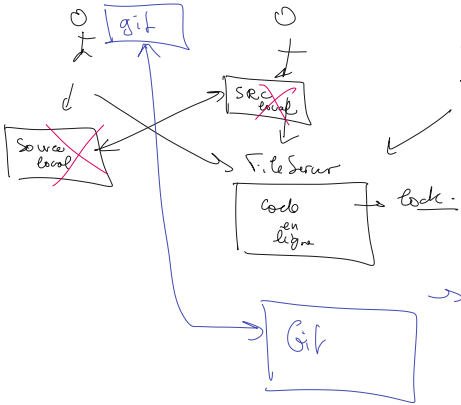


Win32 natif → MFC (natif) → Winform (obsolète)

→ WinUI (3)
(dev natif sur plateforme moderne windows)

Qt
→ Env Dev multiplateforme
Windows / Linux / Android
etc — spécifique —

Git : https://learngitbranching.js.org/?locale=fr_FR



SCM

CVS

Subversion

TFS

Git

SCM (CI)

→ GitLab ici

GitHub Actions

Bit Bucket

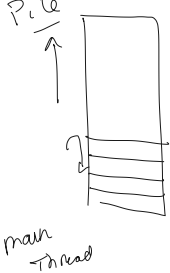
Azure Dev/VisualStudio

etc. ....

Feuille de route - OS

- Compilateur

Pile



+ Unix

Linux OS

main Thread

Per Thread

Jamais de Tableau sur la pile

C/C++

BugGen

POSIX AWS,

→ NFC, Qt, ...

→ boost

→ ~~STL~~ → lib réputée

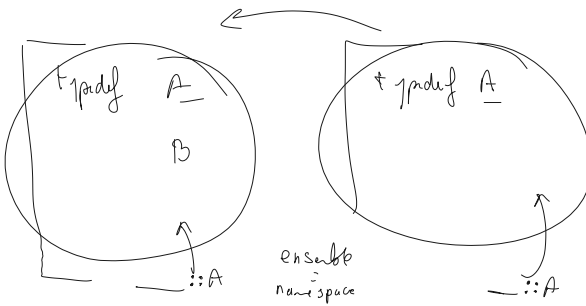
- plus dans les specs C++

STL → Vector

+ SPECS C++

→ STD → Vector (≈ même interface que STL)

→ autres choses -



using namespace ns.

~~ns::~~ Type

libs

boost
NFE
Qt
STL

Specs (officielles)
→ Declarations (Docs)
GCC → implémentation
Microsoft → implémentation

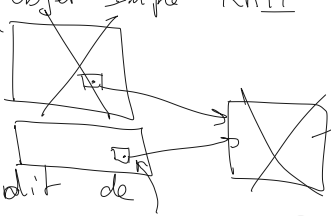
C/c++ de base

malloc / free
new / delete
générateurs de bug

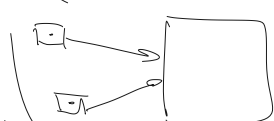
C/c++ "Legacy"
Apprenez / Implémentez le Pattern RAII

C++ Moderne :
RAII est implémenté dès le scope.

Objet Imple RAII unique_ptr



data2 = data



en général ...

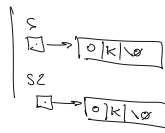
Génération de bug à cause de l'optimisation

C++ legacy.

#define ....

char *s = "ok";

char *s2 = "ok";



memset (s2, "ko").

str

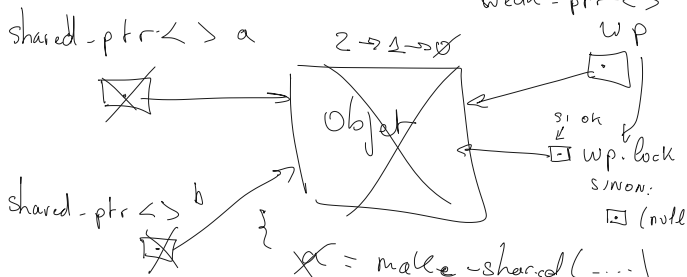
s → "ok"

s2 → "ko"

Release

s = "ko"

s2 = "ko"



x = make_shared(....)

b = a

weak_ptr<> wp;

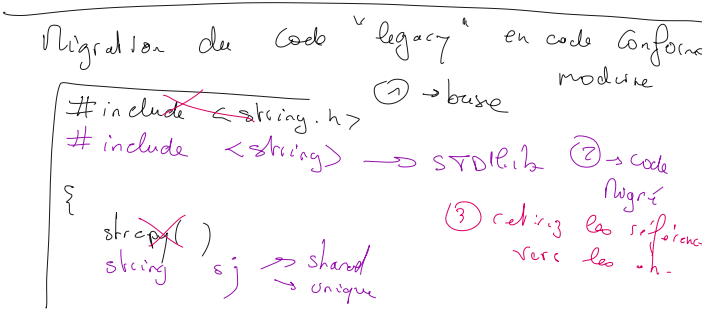
wp = a;

Article recommandé pour comprendre les unique_ptr :
https://learn.microsoft.com/en-us/cpp/cpp/how-to-create-and-use-unique-ptr-instances?view=msvc-170
(user friendly)

Article recommandé pour comprendre les shared_ptr
https://learn.microsoft.com/en-us/cpp/cpp/how-to-create-and-use-shared-ptr-instances?view=msvc-170
(user friendly)

Comprendre les weak_ptr
https://learn.microsoft.com/en-us/cpp/cpp/how-to-create-and-use-weak-ptr-instances?view=msvc-170

Un Weak pointer renverra null au moment de l'appel de son member lock() si l'objet sous-jacent a été libéré. Sinon, vous obtenez l'adresse vers l'objet surveillé.



Lambdas

① Pointeur sur fonction

lien de fonction statique
→ au moment de la compilation

f() → Toujours f()

if (——)
  f1() → lien statique
else
  f2() →

lien / appel dynamique
→ choisi à l'exécution

ptrfunc = &f1;
ptrfunc = &f2;

() → opérateur de branchement avec passage de paramètres
?

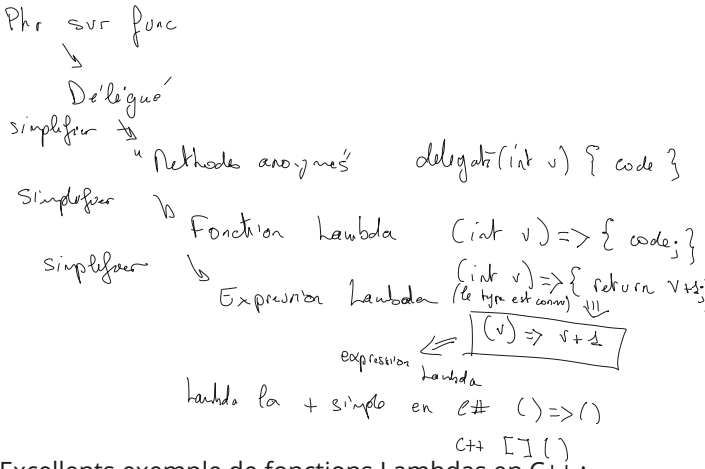
ptrfunc()

⇒ Hétérité

C → C++ → Java → implémenter le pattern observer
                - fonctions anonymes

→ .net (C#, VBnet, C++/CLI) → Délégués

Délégué : classe qui stocke 0..n instances de "Pointeurs sur fonctions"



Excellents exemple de fonctions Lambdas en C++ : https://learn.microsoft.com/fr-fr/cpp/cpp/examples-of-lambda-expressions?view=msvc-160