IC204 - Programation par Objets

Résumé Théorique

12 septembre 2024

Guilherme Nunes Trofino
2022-2024

Table des matières

1. Introduction

Repository Hello ! My name is Guilherme Nunes Trofino and this is my LaTeX notebook of IC204 - Programation par Objets that can be found in my GitHub repository : https://github.com/tr0fin0/classes_ensta.

Disclaimer This notebook is made so it may help others in this subject and is not intend to be used to cheat on tests so use it by your on risk.

Suggestions If you may find something on this document that does not seam correct please reach me by e-mail : guitrofino@gmail.com.

1.1. Informations Matière

Présentation Ce cours sera présenter par M. Jean-Baptiste Laurent qui a pour but d'étudier la Programmation Orienté à Objets : https://perso.ensta-paris.fr/~bmonsuez/Cours et reference externe : https://www.youtube.com/watch?v=vLnPwxZdW4Y.

1.2. run code

Définition C++, et C aussi, est un langage compilé, c'est-à-dire que l'analyse syntaxique et sémantique du programme est réalisée intégralement avant l'éxecution et produition d'un code éxecutable, écrit dans un langage de bas-niveau permetant que l'ordinateur peut directement et efficacement exécuter.

1.2.1. compilator

Définition Converte l'archive source dans une archive de byte code, parfois appeler code objet. Pour compiler un code il faut exécuter le commande suivante :

```
        g++ file.cpp -o executable.exe && ./executable.exe
        g++ file.cpp -o executable.exe  | ./executable.exe
```

Avec ces commandes le code sera d'abbord compilé et après executé.

1.2.2. linker

Définition Après la création d'une archive de byte code le linker va faire reference à d'autres archives necessaires pour le code principale.

1.3. `main`

Définition Inicialization du code se fera toujours avec la function `main` comme démontre :

```
#include <iostream>

int main() {
    std::cout << "main definition" << std::endl;

    return 0;
}
```

La function retournerai 0 si le programme est éxecute sans aucune problème.

1.3.1. `include`

Définition Quand une programme utilise des functions ou classes externes à le code il faut les incluire dans le code principale comme démontré :

```
#include <package>  // syntaxe for packages
#include "class.hpp"// syntaxe for external file
```

C'est important de diviser et séparer les archives du code pour qu'il soit plus facile à comprendre et à l'éditer pendant le dévélopment.

Dans C++ il faut avoir l'extention `.cpp` ou `.hpp` pour convetion.

1.3.2. `functions`

Définition Quand un programme a besoin d'estoquer seulement des functions necessaires pour la `main.cpp` c'est recomende de créer un archive `functions.hpp` et estoquer les informations là dedans pour mieux diviser les archives.

1.4. `types`

1.4.1. `enumerate`

Défition Une enumerate est un nouveau type listant des éléments ne correspondant pas un type particulier. La déclaration la plus simple est :

```
enum enumerate
{
    val1,
    val2,
    ...
};
```

1.4.2. `struct`

Définition Une struct permet de définir un nouveau type de variables regroupant plusieurs variables de types différents. La déclaration la plus simple est :

```
struct struction
{
    int i;
    float j;
    ...
};
```

Struct c'est un type de `class` où tous les méthodes sont publiques par défaut.

1.4.3. `typedef`

Définition Attribuer un alias à un type. La déclaration la plus simple est :

```
typedef unsigned int unit;
```

1.4.4. `template`

Définition Création de functions avec des différents types mais avec la même implementation. La déclaration la plus simple est :

```
template <typename dynamicType> dynamicType function(dynamicType variable)
{
    return variable;
}

template <int> int function(int);
```

Après avoir déclarer la function generic il faut déclarer les functions avec les différents types pour que le compilateur peut savoir comment le faire quand necessaire.

1.5. Mémoire

Définition Langages compilés sont, d'habitude, plus efficaces avec l'usage de mémoire car, la plus part du temps, l'utilisateur doit préciser la quantité de mémoire necessaire.

1.5.1. `pointers`

Définition Variable particulière servant à stocker l'adresse en mémoire centrale d'une variable comme démontré :

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

int main() {
    cout << "pointer arithmetic" << endl << endl;

    int var = 5;
    int* pointVar;
    pointVar = &var;

    cout << "variable" << endl;
    cout << "value:    var = " <<  var << endl;             //  var = 5
    cout << "address: &var = " << &var << endl << endl;     // &var = 0x61fe14


    cout << "pointer" << endl;
    cout << "address, pointVar = " <<  pointVar << endl;    //  pointVar = 0x61fe14
    cout << "value,  *pointVar = " << *pointVar << endl;    // *pointVar = 5

    return 0;
```

1.5.2. Allocation Statique

Définition

1.5.3. Allocation Dynamique

Définition

1.6. C++ 20

Définition C'est necessaire de préciser la bonne version du compilateur parmi le commande suivante :

```
    g++ -std=c++20 file.cpp -o executable.exe && ./executable.exe
    g++ -std=c++2a file.cpp -o executable.exe && ./executable.exe
```

Il y a deux options et il faut essayer lequel marché sur sa machine.

1.6.1. `concept`

Définition Il y a plusieurs manières d'utiliser le concept parmi lequel les deux les plus commans sont :

```cpp
template <typename T>
requires CONDITION
void function(T PARAMETER) {
    ...
}

template <typename T>
void function(T PARAMETER) requires CONDITION {
```

```
9       ...
10  }
```

2. Programation Orientaté à Objets

Définition

2.1. `class`

Définition On considere une implementation generale de `class` comme présenté :

```
#include <iostream>
using std::string;

class Animal
{
public:                         // visible for everyone / everything
    string Name;
    int Age;
    float Weight;

    Animal(string name, int age, float weight)
    {
        Name = name;
        Age = age;
        Weight = weight;
    };


    void print()
    {
        std::cout << "Name:   " << Name   << std::endl;
        std::cout << "Age:    " << Age    << std::endl;
        std::cout << "Weight: " << Weight << std::endl;
    };

protected:                      // visible by certain conditions


private:                        // only visible within the class / object


};
```

Par défaut tout que n'a pas de modificateur de visibilité, `acess modifier` sera considere comme prive.

2.1.1. `objet`

Définition

2.1.2. `constructor`

Définition Function qui inicialize l'objet et doit suivre trois lois :

1. It is a method with no return type ;
2. It has the same name of it's class ;
3. It must be, in most cases, public ;

Le compilateur ira, par défaut, généré une constructeur vide pour qu'un `class` peut être inicialisé mais après inicialisé une constructeur non vide le constructeur vide n'existira plus

2.1.3. `destructor`

Définition

2.2. `encapsulation`

Définition

2.2.1. `public`

Définition

2.2.2. `protected`

Définition

2.2.3. `private`

Définition

2.3. `abstraction`

Définition

2.4. `inheritance`

Définition

2.5. `polymorphism`

Définition

3. Good Practices

3.1. `namespace`

Définition https://www.youtube.com/watch?v=etQX4Mme2f4

4. Travail Dirigé

4.1. Séance 07/09/2022

Présentation Dans ce TD on avait besoin d'implementer une `struct`, qui ressemble plutôt à une `class`, pour voir l'influence du marquer `const` dans le code.

```
#include <stdio.h>

struct counter
{
    int i;
```

```
#include <iostream>
#include "counter.hpp"

void validation(counter &count)
{
```

4.2. Séance 14/09/2022

Présentation Dans ce TD on avait besoin d'implementer deux `class` pour étudier comme les classes sont implementés et comme marché l'heritage entre deux classes.

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

class Counter
{
```

```cpp
#include <iostream>
// #include "Counter.h"
using namespace std;

class Double : public Counter
```

```cpp
#include <iostream>
#include "Counter.hpp"
#include "Double.hpp"
using namespace std;
```

4.3. Séance 21/09/2022

Présentation Dans ce TD on avait besoin d'implementer des `templates` pour étudier comme réduire et améliore le code avec la même implementation mais des types différents.

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


int joinDigits(int base, int newDigit)
```

```cpp
#include <iostream>
#include "join.hpp"
using namespace std;
```

4.4. Séance 28/09/2022

Présentation Dans ce TD on avait besoin d'implementer des `class` avec `template` pour étudier comment les classes peut-être dépendre d'une `type` externe.

```cpp
#include <cstdio>
#include <iostream>


template<typename T = int>  // default template value
```

```cpp
#include <cstdio>
#include <iostream>

typedef size_t size_type;
```

```cpp
#include <iostream>

#include "Number.hpp"
#include "Array.hpp"
#include "Vector.hpp"
```

difference entre array and Vector

4.5. Séance 05/10/2022

Présentation Dans ce TD on avait besoin d'implementer nouvelles functions pour les `operator`'s the une classe.

```cpp
#include <cstdio>
#include <iostream>


template<typename T>    // default template value
```

```cpp
#include <iostream>
#include "Number.hpp"


int main()
```

4.6. Séance 12/10/2022

Présentation Dans ce TD on avait besoin d'étudier C++20 et c'est concepts sur `constraint` et `concept`.

```
#include <iostream>

#pragma once
#include<algorithm>
```

4.7. Séance 19/10/2022

Présentation Dans ce TD on avait besoin d'étudier exceptions et c'est concepts sur `try` et `catch`.

4.8. Séance 26/10/2022

Présentation Dans ce TD on avait besoin d'étudier diamond inheritance et c'est concepts sur `inheritance` et `polymorphisme`.

```
#include <iostream>
#include "counter.hpp"
// #include "decCounter.hpp"
// #include "incCounter.hpp"
// #include "allCounter.hpp"
```

Présentation Utilisation de programmation parallele. matériaux.

```
#include <iostream>
#include <sstream>
#include <string>
#include <chrono>
#include <thread>
```

4.9. Séance 15/11/2022

Présentation Dans ce TD on avait besoin d'essayer des fonctions avec des `threads`. Pour ça on inicialise une número arbitrer de threads qui vont afficher des mensages avec des variables locaux et variables partages/

```
#include <thread>
#include <iostream>

int var = 0;
```

Pour ça c'est necessaire d'ajouter une flag à la compilation :

```
    g++ -std=c++11 -pthread main.cpp -o main.exe | ./main.exe
```

https ://stackoverflow.com/questions/15632198/c11-include-thread-gives-compile-error https ://medium.com/swlh/c-thread-synchronization-at-the-restaurant-ab0d125a0b7b