```
class LeggedRobo(Robot):
    def walk(self):
        self.battery -= 10

if __name__ == "__main__":
    robot = LeggedRobot()
    print(robot.get_battery(), "% remaining")
    robot.work()
    robot.walk()
```

Introduction à Git

01/09/2022

Créer un compte Git + créer une paire de clé cryptographique

+) New Repository : pour créer un projet

Owner = propriétaire du projet / Repository name = nom du projet (sans accents et espaces)
Organisation

Ajouter un README pour décrire le projet au format Markdown.

Choisir une licence BSD 2-Clause

Un projet vide contient : -> une licence
-> un README

Commit (modification des fichiers ou de l'arborescence)

↳ conseil : expliquer en quelques lignes la raison du commit

-> Browse permet de voir l'état du projet au moment du commit.

Fichier : Séparer les headers (contenant les prototypes publics) .hpp et la description fonctions.cpp dans 2 fichiers distincts.

Remarque :

```
#ifndef CONCEPTION_ORIENTEE_OBJET_EXAMPLE_ADDER_HPP
#define CONCEPTION_ORIENTEE_OBJET_EXAMPLE_ADDER_HPP
// déclaration des prototypes
#endif
```

Permet de s'assurer que le compilateur n'inclut qu'une seule fois la bibliothèque en tête du fichier

Fichier CMake : CMakeLists.txt permet de gérer la compilation de tous les fichiers dans leur dernière version, dans le dossier src

```
cmake_minimum_required (VERSION 3.18)

project(
    Conception_orientee_Objet
    VERSION 0.1.0 // choisir du semantic versioning
    DESCRIPTION "exemple"
    HOME_PAGE_URL "https://"
    LANGAGES CXX
)

add_library(
    exemple_adder SHARED include/conception...) // chemin des librairies

target_include_directories(
    exemple_adder PUBLIC $<BUILD_INTERFACE
    :${CMAKE_SOURCE_... // chemin d'accès des fichiers sources

add_executable(src/adder.cpp)
target_link_libraries(adder PUBLIC exemple_adder)
// réalise un lien entre les 2 codes
```

Utilisation depuis un terminal en local

↳ cmake -S . -B build
↑ création d'un dossier build où CMake place les fichiers temporaires

Compilation avec CMake :

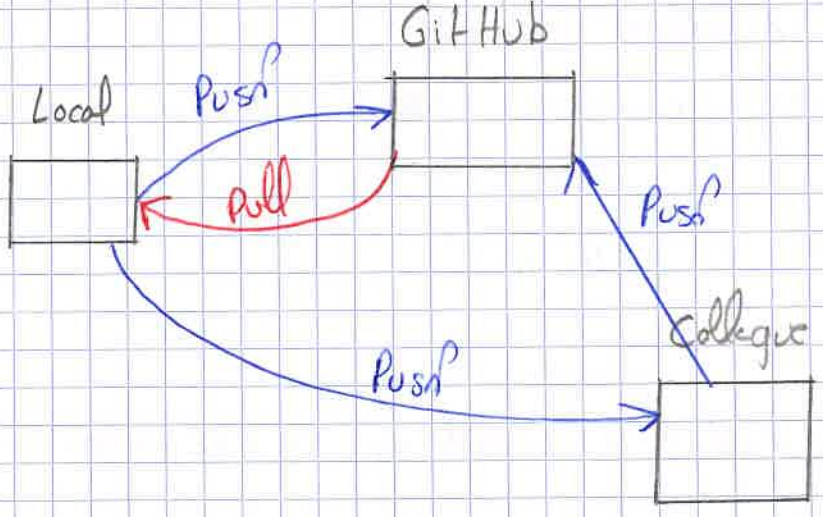
↳ cmake --build build
↑ nom du fichier où se trouve les fichiers temporaires

Librairie dynamique

en résulte un .so dans build

et l'exécutable adder.
Le fichier adder.cpp contient les fonctions (inclut le CMakeLists.txt)

Remarque : il est possible de récupérer les arguments passer en appel du programme.



Pour qu'un répertoire local puisse se synchroniser avec Git, il doit contenir un fichier .git.

La commande ldd permet de lister toutes les librairies dont dépend l'exécutable passé en paramètre.

Ajout d'un test.

Le répertoire tests contient un CMakeLists.txt doit contenir include(CTest) et la possibilité de ne pas compiler le test if(BUILD_TESTING)
add_subdirectory(tests)
endif()

On donne la possibilité d'exécuter la commande add_executable(test-add test-add.cpp) add_test(NAME test-add COMMAND test-add)
target_link_libraries(test-add PUBLIC example-adder) //Lien de la librairie
Le code de test contient: #include <cassert>

inclut le bibliothèque vue précédemment #include "conception-orientee-objet/example-adder.hpp"

```
auto main() -> int {
    assert(coo::add(1,2) == 3);
    return 0;
}
```

Dans le terminal on lance la compilation avec ce nouveau commit se fait avec cmake --build build -t test
-t test: target vers les tests (facultatif => evite de rebuild tout le projet)

cmake -B build -DBUILD_TEST=OFF pour ne pas compiler les test

Ajout de biding Python en C++

Dans le dossier Python,

Etape 1: Définir une option CMake option(BUILD_PYTHON_INTERFACE "Build the python biding ON")

Etape 2: s'assurer que Python soit installé sur l'ordinateur

```
if(BUILD_PYTHON_INTERFACE)
    find_package(Python REQUIRED COMPONENTS Interpreter Development.Module)
    find_package(Boost REQUIRED COMPONENTS python)
    add_subdirectory(python)
endif()
```

Etape 3 : Lier les librairies

ajoute l'interpréteur dans le nom du .so résultant

python_add_library (conception_orientee_objet MODULE WITH_SOABI
conception_orientee_objet.cpp)
target_link_libraries (conception_orientee_objet PUBLIC
example_adder
Boost::python)

Etape 4 : Ecriture des fonctions dans le fichier .cpp

```
#include <boost/python.hpp>
#include "conception_orientee_objet/example_adder.hpp"
BOOST_PYTHON_MODULE(conception_orientee_objet) { boost::python::def
("add", coo::add); }
```

url GitHub github.com/nim65s/conception_orientee_objet