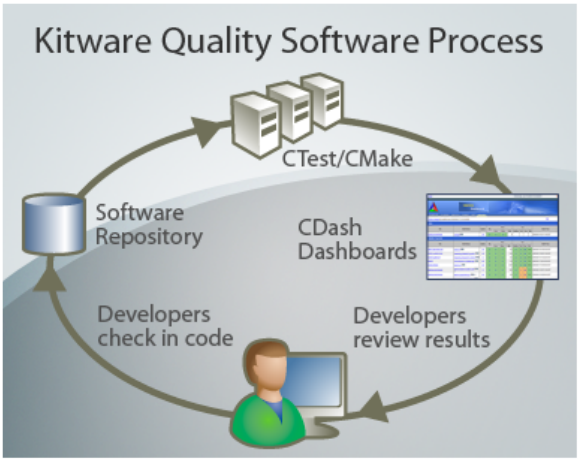
Introduction à CMake

CMake fournit un ensemble d'outils permettant de compiler un projet pour différentes plateformes, de faire des tests et de créer des packages pour différents systèmes. Il est utilisé dans de nombreux projets tels que KDE et MySQL et est une bonne alternative aux autotools. Lorsque vous installez **CMake** sur votre système, vous avez de base les commandes suivantes:

- **cmake**: création de Makefile et compilation de projets
- **ctest**: mise en place de tests sur vos projets
- **cpack**: création de package

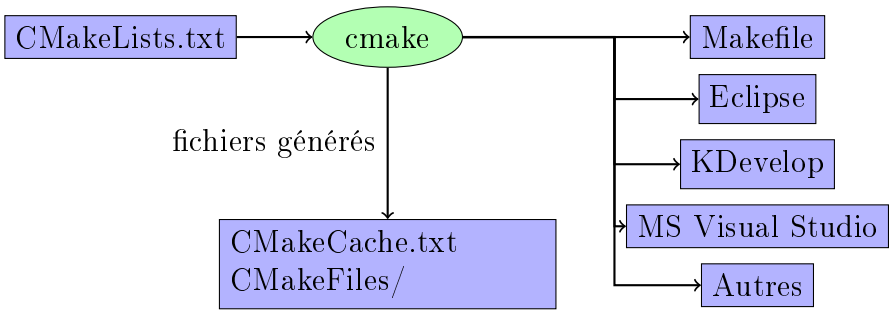
On peut également citer **CDash** qui fournit une interface web donnant toutes les informations sur les tests soumis sur différentes plateformes. Pour plus d'informations, vous pouvez vous reporter à cette page http://www.cdash.org/.
On peut résumer le processus d'un logiciel utilisant CMake par la figure suivante disponible sur la page de CMake:



Au cours de ce TP, nous n'aborderons que **CMake** et **CTest**.
CMake a son langage propre. La syntaxe est la suivante:

- `# Ceci est un commentaire`
- `command(argument1 argument2 ... argumentN)`
- `COMMAND(argument1 argument2 ... argumentN)`

Nous verrons dans la suite les principales commandes pour construire la plupart des projets. Le fichier source, comportant tous les paramètres du projet et la démarche à suivre pour le construire, se nomme ***CMakeLists.txt***. La procédure de construction peut être résumée par la figure suivante



Le fichier ***CMakeCache.txt*** définit un ensemble de variables de configuration et d'informations sur votre système. Le répertoire ***CMakeFiles*** contient les fichiers intermédiaires de compilation, des fichiers temporaires et autres fichiers de configuration qui ne regardent que **CMake**. On ne s'intéressera dans la suite qu'à la construction de ***Makefile*** qui est celle par défaut. Pour construire d'autres projets, vous devez spécifier votre générateur à l'aide de l'option `-G`. Pour plus d'information, vous pouvez vous reporter à la page suivante générateurs de CMake.

1 Création d'un projet simple

On va commencer par écrire notre premier ***CMakeLists.txt***. Pour ce faire, créez un répertoire ***hello*** dans lequel vous mettrez l'incontournable ***Hello world!!***.
Programme ***hello.c***:

```
#include <stdio.h>

int main()
{
  printf("hello world!!\n");
  return 0;
}
```

Le première commande que nous allons voir ici est ***add_executable*** permettant d'ajouter un exécutable à notre projet. La syntaxe est la suivante

```
add_executable(<name> source1 source2 ... sourceN)
```

où

- ***name*** est bien entendu le nom de l'exécutable,
- ***sourcex*** sont les fichiers sources permettant de construire l'exécutable.

Il existe d'autres options qui ne seront pas développées ici mais que vous trouverez aisément dans la documentation de **CMake**.

1. Créez le fichier ***CMakeLists.txt*** au même niveau que votre programme et ajoutez y la commande ***add_executable*** avec les bons arguments.
2. Créez un répertoire ***build*** dans lequel vous allez construire votre projet. Il est d'usage de construire le projet dans un autre répertoire que le répertoire source. Cela permet d'effacer la construction facilement et de ne pas mélanger les sources et la construction.
3. Allez dans le répertoire ***build*** puis tapez la commande

   ```
   terminal$ cmake ..
   ```

   Cette commande permet (comme expliqué précédemment) de créer le fichier ***CMakeCache.txt***, le répertoire ***CMakeFiles*** et le ***Makefile***.
4. tapez ensuite

   ```
   terminal$ make
   ```

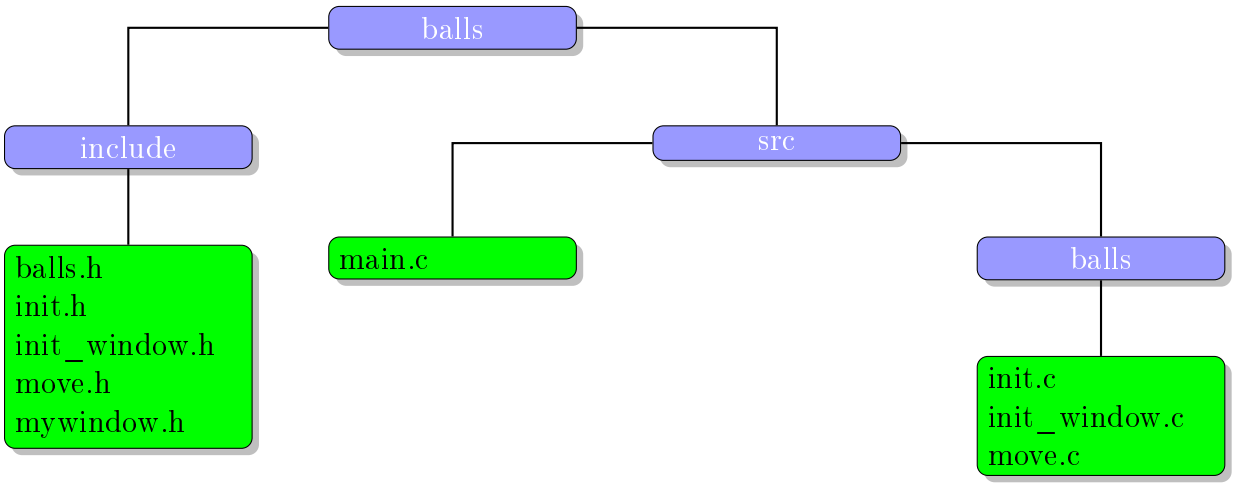
Si tout s'est bien déroulé, vous devriez avoir votre exécutable dans le répertoire ***build*** que vous pouvez exécuter.
Comme vous avez pu le constater, les commandes ***cmake*** et ***make*** ne sont pas très bavardes et il peut être intéressant de savoir ce qui est réellement fait derrière. Pour cela, vous pouvez utiliser les commandes suivantes

```
terminal$ cmake --trace ..
terminal$ make VERBOSE=1
```

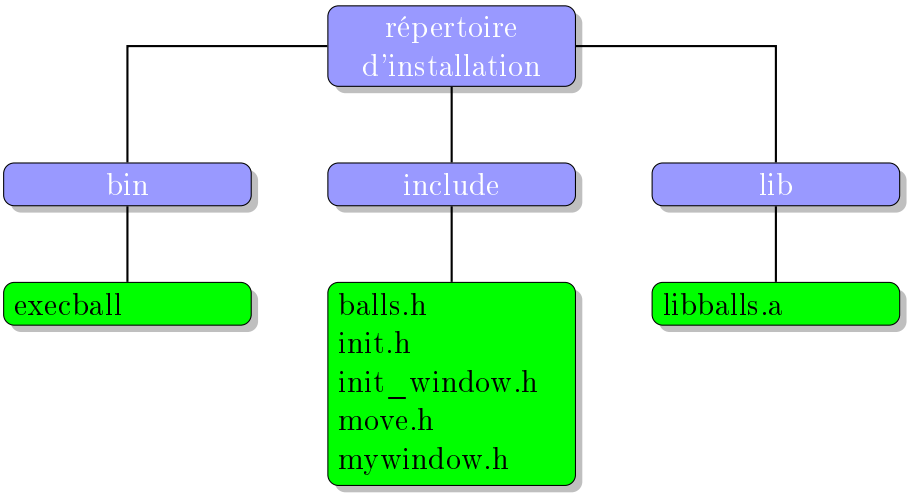
N'oubliez pas de faire avant un `make clean` pour que le projet soit entièrement recompilé et voir les commandes du compilateur.

2 Création d'un projet écrit en langage C

Nous allons voir à présent comment compiler un projet un peu plus complexe. Dans cette exemple, les principales commandes de **CMake** seront abordées.
Nous avons mis à votre disposition un projet balls qui a l'arborescence suivante



Dans ce projet, nous créons une balle que nous déplaçons et qui rebondit sur les bords de la fenêtre. Nous allons vous guider pas à pas, pour qu'à la fin, votre projet construise l'arborescence suivante



Nous allons dans un premier temps nommer notre projet via la commande ***project***. La syntaxe est la suivante

```
project(<projectname> [languageName1 languageName2 ... ])
```

où

- ***projectname*** est le nom du projet,
- ***languageName*** est le nom du langage utilisé dans le projet.

Par défaut le C et le C++ sont reconnus. Donc, dans notre cas, nous n'avons rien à ajouter. Cette commande permet de définir les variables

- ***<projectName>_BINARY_DIR*** qui représente le répertoire où le projet sera compilé,
- ***<projectName>_SOURCE_DIR*** qui représente le répertoire où sont les sources.

Vous pouvez afficher à tout moment des messages lorsque vous exécutez la commande cmake en ajoutant dans votre ***CMakeLists.txt*** la commande

```
message( [STATUS|WARNING|AUTHOR_WARNING|FATAL_ERROR|SEND_ERROR ] "message to display" ...)
```

Créez un ***CMakeList.txt*** dans le répertoire ***balls*** en y mettant les lignes

```
project(boule)

message("rep source du projet: ${boule_SOURCE_DIR}")
message("rep compilation du projet: ${boule_BINARY_DIR}")
```

Puis créez un répertoire ***build*** dans lequel vous taperez la commande

```
terminal$ cmake ..
```

Vous pouvez également passer en mode interface en utilisant la commande ***ccmake***.

```
terminal$ ccmake ..
```

Puis tapez sur `t`, vous aurez un ensemble d'informations sur les variables déjà existantes.

Nous allons à présent ajouter nos sous-répertoires via la commande

```
add_subdirectory(source_dir )
```

Ajoutez le répertoire ***include*** et le répertoire ***src***. Relancez la commande ***cmake***. Que se passe-t-il ?
Ajoutez la commande `cmake_minimum_required(VERSION 2.8)` dans votre ***CMakeLists.txt***.

Comme vous avez pu le constater, il nous faut également un fichier ***CMakeLists.txt*** dans les répertoires ***include*** et ***src***.
Nous allons commencer par le répertoire ***include***. Nous voulons ajouter à notre projet tous les fichiers headers (`.h`) et les mettre dans le répertoire ***include*** du répertoire d'installation du projet. Il faut dans un premier temps lister tous les `.h`. Vous pourriez le faire à la main mais lorsque vous avez une vingtaine de fichiers, ce n'est pas forcément pratique. Vous pouvez utiliser la commande ***file*** qui permet de faire des opérations sur les fichiers. La commande est la suivante

```
file(GLOB variable [RELATIVE path] [globbing expressions]...)
```

où

- ***GLOB*** génère la liste des fichiers qui correspondent à la globbing expressions
- ***variable*** est le nom que l'on donne à cette liste
- ***RELATIVE path*** est le chemin où on va chercher la globbing expressions

Dans notre cas, nous pouvons mettre

```
file(GLOB include_H . *.h)
```

Il ne reste plus qu'à spécifier le répertoire d'installation via la commande

```
install(files <files to add> destination <dir>)
```

Dans notre cas, nous pouvons mettre

```
INSTALL(FILES ${include_H} DESTINATION include)
```

Nous en avons terminé avec le répertoire ***include***. Passons au répertoire ***src***. Pour compiler tous les fichiers C, nous allons avoir besoin de ces fichiers headers. Nous devons donc le spécifier à **CMake** via la commande

```
include_directories(dir1 dir2 ...)
```

Ajoutez dans le ***CMakeLists.txt*** se trouvant dans le répertoire ***balls***

```
include_directories(include)
```

Créez un fichier ***CMakeLists.txt*** dans le répertoire ***src*** puis dites à **CMake** d'ajouter le répertoire ***balls*** se trouvant dans ***src***. Allez dans le répertoire ***src/balls*** et créez un ***CMakeLists.txt*** dans lequel vous listerez tous les fichiers C que vous mettrez dans une variable ***SRC***.
A partir de cette liste, nous allons créer une librairie via la commande

```
add_library(<name> [STATIC | SHARED | MODULE]
            source1 source2 ... sourceN)
```

Dans notre cas, nous aurons

```
ADD_LIBRARY(balls ${SRC})
```

Ajoutez à présent

```
install(TARGETS balls DESTINATION lib)
```

pour avoir ***libballs.a*** dans le répertoire ***lib*** du répertoire d'installation.
Nous revenons au ***CMakeLists.txt*** se trouvant dans ***src*** pour créer un exécutable à partir du fichier ***main.c*** et de la librairie créée précédemment.
Créez une variable via la commande

```
SET(PROJECT_LIBS balls)
```

Puis ajoutez votre exécutable via la commande ***add_executable*** vue précédemment

```
add_executable(execballs main.c)
```

Il ne reste plus qu'à utiliser la commande

```
target_link_libraries(<target> [item1 [item2 [...]]])
```

pour l'édition des liens. Nous ajoutons également notre exécutable dans le répertoire ***bin*** du répertoire d'installation. Dans notre cas, nous avons

```
TARGET_LINK_LIBRARIES(execballs ${PROJECT_LIBS})
install(TARGETS execballs DESTINATION bin)
```

Vous pouvez à présent lancer ***cmake*** ou ***ccmake***. Pour spécifier le répertoire d'installation, vous avez 2 possibilités

- soit vous modifiez la variable `CMAKE_INSTALL_PREFIX` via cmake en utilisant la commande

  ```
  terminal$ cmake -DCMAKE_INSTALL_PREFIX=/la/ou/je/veux ..
  ```

- soit vous lancez la commande ***ccmake*** et vous modifiez la variable `CMAKE_INSTALL_PREFIX`.

Tapez ensuite `make install`. Allez dans le répertoire d'installation, puis tapez `./bin/execballs`.

3 Création d'un projet multi-langages utilisant des bibliothèques tierces

Nous reprenons ici les sources du projet précédent dans lequel nous ajoutons un programme Fortran qui sera appelé dans un programme C++. Nous ajoutons également une interface graphique réalisée à partir de **Qt** pour voir nos boules. Le répertoire source s'appelle ***ballsWithGui***.
Lancez ***cmake***, puis installez le projet dans, par exemple, ***"/home/lem2i/installBallsWithGui"***.
Parcourez l'arborescence et regardez ce qui a changé.

La commande ***find_package*** permet de chercher des programmes sur votre système et les informations relatives (comme par exemple le répertoire des include ou des librairies).
CMake fournit un ensemble de modules avec l'extension ***.cmake*** permettant de trouver certains programmes sur votre système. Les fichiers se trouvent dans ***"/usr/share/cmake-2.8/Modules"***.
Vous pouvez également créer vos propres modules mais nous ne le verrons pas ici.

4 Création de tests et génération de documentation

Nous avons rajouté quelques tests ainsi que la génération de documentation via **doxygen** dans le projet ***ballsWithGuiFinal***.
Pour rendre les tests possible, il suffit d'ajouter ***enable_testing()*** puis d'ajouter les tests en

utilisant la commande

```
add_test(testname Exename arg1 arg2 ...)
```

Regardez le ***CMakeLists.txt*** dans le répertoire ***tests***. Pour lancer les tests, il suffit de taper `make test` après avoir fait `make`.
La documentation est créée automatiquement si **doxygen** est sur le système et si ***INSTALL_DOC*** est à `ON`. Après avoir fait `make install`, allez dans le répertoire ***doc/html*** se trouvant dans le répertoire d'installation.