Cpp concept project

Generated by Doxygen 1.8.20

Chapter 1

C++ concepts project

1.1 Idea

This project serves as sample/concept project for further projects :thumbsup:

1.2 Related documents

- `Notes`
- Markdown cheatsheet
- Project structure
- Unit testing

1.3 Structure

1.3.1 Folders

- **bin**: output executables go here (for the app, tests and spikes)
- **build**: containing all the object files (removed by clean)
- **doc**: documentation files
- **ideas**: smaller classes or files to test technologies or ideas
- **include**: all project header files, all necessary third-party header files (which are not in /usr/local/include)
- **lib**: any library that get compiled by the project, third party or any needed in development
- **resources**: resources
- **src**: the application and application's source files
- **test**: all test code files

1.4 Content (Concepts)

1.4.1 Programming concepts

- Classes
  - Inheritance
- Templates
- ...

1.4.2 Documentation

The documentation is intrinsically implemented using `doxygen`. In order to do that:

- specify path to doxygen binary in the Makefile
- execute *make doc*

The *README.md* file is used for the Mainpage of the documentation. Set the settings for doxygen in *doc/Doxyfile*.

1.4.3 Makefile

Following targets are implemented:

- **all** default make
- **remake**
- **clean**
- **cleaner**
- **resources**
- **sources**
- **directories**
- **ideas**
- **tester**
- **doc**

Chapter 2

CMake

2.1 Links

- Repository
- Awesome-CMake list

2.1.1 Documentation

- CMake official documentation
- The Architecture of Open Source Applications

2.1.2 Tutorials & Instructions

- Effective Modern CMake (Dos & Don'ts)
- GitBook: Introduction to Modern CMake
- CMake Cookbook
- CMake Primer

2.1.3 Videos

- Intro to CMake
- Using Modern CMake Patterns to Enforce a Good Modular Design
- Effective CMake
- Embracing Modern CMake

2.2 Basics

2.2.1 CMake Version

```
#minmum CMake version
cmake_minimum_required(VERSION 3.12)
if(${CMAKE_VERSION} VERSION_LESS 3.12)
    cmake_policy(VERSION ${CMAKE_MAJOR_VERSION}.${CMAKE_MINOR_VERSION})
endif()
```

2.2.2 VARIABLES

```
# Local variable
set(MY_VARIABLE "value")
set(MY_LIST "one" "two")
# Cache variable
set(MY_CACHE_VARIABLE "VALUE" CACHE STRING "Description")
# Environmental variables
set(ENV{variable_name} value) #access via $ENV{variable_name}
```

2.2.3 PROPERTIES

```
set_property(TARGET TargetName PROPERTY CXX_STANDARD 11)
set_target_properties(TargetName PROPERTIES CXX_STANDARD 11)
get_property(ResultVariable TARGET TargetName PROPERTY CXX_STANDARD)
```

2.2.4 Output folders

```
# set output folders
set(PROJECT_SOURCE_DIR)
set(CMAKE_SOURCE_DIR ...)
set(CMAKE_BINARY_DIR ${CMAKE_SOURCE_DIR}$/bin)
set(EXECUTABLE_OUTPUT_PATH ${CMAKE_BINARY_DIR})
set(LIBRARY_OUTPUT_PATH ${CMAKE_BINARY_DIR})
```

2.2.5 Sources

```
# set sources
set(SOURCES example.cu)
file(GLOB SOURCES *.cu)
```

2.2.6 Executables & targets

Add executable/create target:

```
#add_executable(example ${PROJECT_SOURCE_DIR}/example.cu)
add_executable(miluphcuda ${SOURCES})
# add include directory to target
target_include_directories(miluphcdua PUBLIC include) #PUBLIC/PRIVATE/INTERFACE
# add compile feature to target
target_compile_features(miluphcuda PUBLIC cxx_std_11)
# chain targets (assume "another" is a target)
add_library(another STATIC another.cpp another.h)
target_link_libraries(another PUBLIC miluphcuda)
```

2.2.7 PROGRAMMING IN CMAKE

Keywords:

- NOT
- TARGET
- EXISTS
- DEFINED
- STREQUAL
- AND
- OR
- MATCHES
- ...

2.2.7.1 Control flow

```
if(variable)
    # If variable is `ON`, `YES`, `TRUE`, `Y`, or non zero number
else()
    # If variable is `0`, `OFF`, `NO`, `FALSE`, `N`, `IGNORE`, `NOTFOUND`, `""`, or ends in `-NOTFOUND`
#endif()
```

2.2.7.2 Loops

- `foreach(var IN ITEMS foo bar baz) ...`
- `foreach(var IN LISTS my_list) ...`
- `` `foreach(var IN LISTS my_list ITEMS foo bar baz) ... ``

2.2.7.3 Generator expression

```
target_include_directories(
        MyTarget
        PUBLIC
        $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
        $<INSTALL_INTERFACE:include>
)
```

2.2.7.4 Functions (& macros)

```
function(SIMPLE REQUIRED_ARG)
    message(STATUS "Simple arguments: ${REQUIRED_ARG}, followed by ${ARGV}")
    set(${REQUIRED_ARG} "From SIMPLE" PARENT_SCOPE)
endfunction()
simple(This)
message("Output: ${This}")
```

2.2.8 COMMUNICATION WITH CODE

2.2.8.1 Configure File

```
configure_file()
...
```

2.2.8.2 Reading files

...

2.2.9 RUNNING OTHER PROGRAMS

2.2.9.1 command at configure time

```
find_package(Git QUIET)
if(GIT_FOUND AND EXISTS "${PROJECT_SOURCE_DIR}/.git")
    execute_process(COMMAND ${GIT_EXECUTABLE} submodule update --init --recursive
            WORKING_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}
            RESULT_VARIABLE GIT_SUBMOD_RESULT)
    if(NOT GIT_SUBMOD_RESULT EQUAL "0")
        message(FATAL_ERROR "git submodule update --init failed with ${GIT_SUBMOD_RESULT}, please checkout
       submodules")
    endif()
endif()
```

2.2.9.2 command at build time

```
find_package(PythonInterp REQUIRED)
add_custom_command(OUTPUT "${CMAKE_CURRENT_BINARY_DIR}/include/Generated.hpp"
        COMMAND "${PYTHON_EXECUTABLE}" "${CMAKE_CURRENT_SOURCE_DIR}/scripts/GenerateHeader.py" --argument
        DEPENDS some_target)
add_custom_target(generate_header ALL
        DEPENDS "${CMAKE_CURRENT_BINARY_DIR}/include/Generated.hpp")
install(FILES ${CMAKE_CURRENT_BINARY_DIR}/include/Generated.hpp DESTINATION include)
```

2.3 Libraries

```
# make a library
add_library(one STATIC two.cpp three.h) # STATIC/SHARED/MODULE
```

2.4 Language/Package related

2.4.1 C

```
# set C compiler
set(CMAKE_C_COMPILER "/usr/bin/gcc-7")
execute_process (
        COMMAND bash -c "git describe --abbrev=4 --dirty --always --tags'"
        OUTPUT_VARIABLE GIT_VERSION
)
#set(CMAKE_BUILD_TYPE DEBUG)
set(CMAKE_C_FLAGS "-c -std=c99 -O3 -DVERSION=\"${GIT_VERSION})\" -fPIC")
#set(CMAKE_C_FLAGS_DEBUG "-O0 -ggdb")
#set(CMAKE_C_FLAGS_RELEASE "-O0 -ggdb")
```

2.4.2 C++

...

2.4.3 CUDA

See Combining CUDA and Modern CMake

2.4.3.1 Enable Cuda support

CUDA is not optional
```
project(MY_PROJECT LANGUAGES CUDA CXX)
```

CUDA is optional
```
enable_language(CUDA)
```

Check whether CUDA is available
```
include(CheckLanguage)
check_language(CUDA)
```

2.4.3.2 CUDA Variables

Exchange *CXX* with *CUDA*

E.g. setting CUDA standard:
```
if(NOT DEFINED CMAKE_CUDA_STANDARD)
    set(CMAKE_CUDA_STANDARD 11)
    set(CMAKE_CUDA_STANDARD_REQUIRED ON)
endif()
```

2.4.3.3 Adding libraries / executables

As long as ∗.cu∗ is used for CUDA files, the procedure is as normal.

With separable compilation
```
set_target_properties(mylib PROPERTIES
                            CUDA_SEPARABLE_COMPILATION ON)
```

2.4.3.4 Architecture

Use `CMAKE_CUDA_ARCHITECTURES` variable and the `CUDA_ARCHITECTURES property` on targets.

2.4.3.5 Working with targets

Compiler option
```
"$<$<BUILD_INTERFACE:$<COMPILE_LANGUAGE:CXX>:-fopenmp>$<$<BUILD_INTERFACE:$<COMPILE_LANGUAGE:CUDA>:-Xcompiler=-fopenmp>"
```

Use a function that will fix a C++ only target by wrapping the flags if using a CUDA compiler
```
function(CUDA_CONVERT_FLAGS EXISTING_TARGET)
    get_property(old_flags TARGET ${EXISTING_TARGET} PROPERTY INTERFACE_COMPILE_OPTIONS)
    if(NOT "${old_flags}" STREQUAL "")
        string(REPLACE ";" "," CUDA_flags "${old_flags}")
        set_property(TARGET ${EXISTING_TARGET} PROPERTY INTERFACE_COMPILE_OPTIONS

       "$<$<BUILD_INTERFACE:$<COMPILE_LANGUAGE:CXX>:${old_flags}>$<$<BUILD_INTERFACE:$<COMPILE_LANGUAGE:CUDA>:-Xcompiler=${CU
            )
    endif()
endfunction()
```

2.4.3.6 Useful variables

- `CMAKE_CUDA_TOOLKIT_INCLUDE_DIRECTORIES`: Place for built-in Thrust, etc
- `CMAKE_CUDA_COMPILER`: NVCC with location

```
set(CUDA_DIR "/usr/local/cuda-10.0")
set(CMAKE_CUDA_COMPILER ${CUDA_DIR}/bin/nvcc)
set(CMAKE_CUDA_FLAGS "-ccbin ${CC} -x cu -c -dc -O3  -Xcompiler "-O3 -pthread" -Wno-deprecated-gpu-targets
      -DVERSION=\"${GIT_VERSION}\"  --ptxas-options=-v")
set(CMAKE_CUDA_FLAGS_DEBUG ...)
set(CMAKE_CUDA_HOST_COMPILER ...)
set(CMAKE_CUDA_EXTENSIONS ...)
set(CMAKE_CUDA_STANDARD ...)
set(CMAKE_CUDA_RUNTIME_LIBRARY ...)
...
```

2.4.4 OpenMP

2.4.4.1 Enable OpenMP support

```
find_package(OpenMP)
if(OpenMP_CXX_FOUND)
    target_link_libraries(MyTarget PUBLIC OpenMP::OpenMP_CXX)
endif()
```

2.4.5 Boost

The Boost library is included in the find packages that CMake provides.

(Common) Settings related to boost

- `set(Boost_USE_STATIC_LIBS OFF)`
- `set(Boost_USE_MULTITHREADED ON)`
- `` `set(Boost_USE_STATIC_RUNTIME OFF) ``

E.g.: using the `Boost::filesystem` library

```
set(Boost_USE_STATIC_LIBS OFF)
set(Boost_USE_MULTITHREADED ON)
set(Boost_USE_STATIC_RUNTIME OFF)
find_package(Boost 1.50 REQUIRED COMPONENTS filesystem)
message(STATUS "Boost version: ${Boost_VERSION}")
# This is needed if your Boost version is newer than your CMake version
# or if you have an old version of CMake (<3.5)
if(NOT TARGET Boost::filesystem)
    add_library(Boost::filesystem IMPORTED INTERFACE)
    set_property(TARGET Boost::filesystem PROPERTY
        INTERFACE_INCLUDE_DIRECTORIES ${Boost_INCLUDE_DIR})
    set_property(TARGET Boost::filesystem PROPERTY
        INTERFACE_LINK_LIBRARIES ${Boost_LIBRARIES})
endif()
```

2.4.6 MPI

2.4.6.1 Enable MPI support

```
find_package(MPI REQUIRED)
message(STATUS "Run: ${MPIEXEC} ${MPIEXEC_NUMPROC_FLAG} ${MPIEXEC_MAX_NUMPROCS} ${MPIEXEC_PREFLAGS}
       EXECUTABLE ${MPIEXEC_POSTFLAGS} ARGS")
target_link_libraries(MyTarget PUBLIC MPI::MPI_CXX)
```

2.5 Adding features

2.5.1 Set default build type

```
set(default_build_type "Release")
if(NOT CMAKE_BUILD_TYPE AND NOT CMAKE_CONFIGURATION_TYPES)
  message(STATUS "Setting build type to '${default_build_type}' as none was specified.")
  set(CMAKE_BUILD_TYPE "${default_build_type}" CACHE
      STRING "Choose the type of build." FORCE)
  # Set the possible values of build type for cmake-gui
  set_property(CACHE CMAKE_BUILD_TYPE PROPERTY STRINGS
    "Debug" "Release" "MinSizeRel" "RelWithDebInfo")
endif()
```

2.5.2 Meta compiler features

```
target_compile_features(myTarget PUBLIC cxx_std_11)
set_target_properties(myTarget PROPERTIES CXX_EXTENSIONS OFF)
set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_CXX_EXTENSIONS OFF)
set_target_properties(myTarget PROPERTIES
    CXX_STANDARD 11
    CXX_STANDARD_REQUIRED YES
    CXX_EXTENSIONS NO
)
```

2.5.3 Position independent code (-fPIC)

```
set(CMAKE_POSITION_INDEPENDENT_CODE ON)
# or target dependent
set_target_properties(lib1 PROPERTIES POSITION_INDEPENDENT_CODE ON)
```

2.5.4 Little libraries

```
find_library(MATH_LIBRARY m)
if(MATH_LIBRARY)
    target_link_libraries(MyTarget PUBLIC ${MATH_LIBRARY})
endif()
```

2.5.5 Modules

2.5.5.1 CMakeDependentOption

```
include(CMakeDependentOption)
cmake_dependent_option(BUILD_TESTS "Build your tests" ON "VAL1;VAL2" OFF)
```

which is equivalent to

```
if(VAL1 AND VAL2)
    set(BUILD_TESTS_DEFAULT ON)
else()
    set(BUILD_TESTS_DEFAULT OFF)
endif()
option(BUILD_TESTS "Build your tests" ${BUILD_TESTS_DEFAULT})
if(NOT BUILD_TESTS_DEFAULT)
    mark_as_advanced(BUILD_TESTS)
endif()
```

2.5.5.2 CMakePrintHelpers

```
cmake_print_properties
cmake_print_variables
```

2.5.5.3 CheckCXXCompilerFlag

Check whether flag is supported

```
include(CheckCXXCompilerFlag)
check_cxx_compiler_flag(-someflag OUTPUT_VARIABLE)
```

2.5.5.4 WriteCompilerDetectionHeader

Look for a list of features that some compilers support and write out a C++ header file that lets you know whether that feature is available

```
write_compiler_detection_header(
  FILE myoutput.h
  PREFIX My
  COMPILERS GNU Clang MSVC Intel
  FEATURES cxx_variadic_templates
)
```

2.5.5.5 try_compile / try_run

```
try_compile(
  RESULT_VAR
    bindir
  SOURCES
    source.cpp
)
```

2.6 Debugging

2.6.1 Printing variables

```
message(STATUS "MY_VARIABLE=${MY_VARIABLE}")
# or using module
include(CMakePrintHelpers)
cmake_print_variables(MY_VARIABLE)
cmake_print_properties(
    TARGETS my_target
    PROPERTIES POSITION_INDEPENDENT_CODE
)
```

2.6.2 Tracing a run

```
cmake -S . -B build --trace-source=CMakeLists.txt #--trace-expand
```

2.7 Including projects

2.7.1 Fetch

E.g.: download Catch2

```
FetchContent_Declare(
  catch
  GIT_REPOSITORY https://github.com/catchorg/Catch2.git
  GIT_TAG        v2.13.0
)
# CMake 3.14+
FetchContent_MakeAvailable(catch)
```

2.8 Testing

2.8.1 General

Enable testing and set a BUILD_TESTING option
```
if(CMAKE_PROJECT_NAME STREQUAL PROJECT_NAME)
    include(CTest)
endif()
```

Add test folder
```
if(CMAKE_PROJECT_NAME STREQUAL PROJECT_NAME AND BUILD_TESTING)
    add_subdirectory(tests)
endif()
```

Register targets
```
add_test(NAME TestName COMMAND TargetName)
add_test(NAME TestName COMMAND $<TARGET_FILE:${TESTNAME}>)
```

2.8.2 Building as part of the test

```
add_test(
  NAME
    ExampleCMakeBuild
  COMMAND
    "${CMAKE_CTEST_COMMAND}"
              --build-and-test "${My_SOURCE_DIR}/examples/simple"
                               "${CMAKE_CURRENT_BINARY_DIR}/simple"
              --build-generator "${CMAKE_GENERATOR}"
              --test-command "${CMAKE_CTEST_COMMAND}"
)
```

2.8.3 Testing frameworks

2.8.3.1 GoogleTest

See Modern CMake: GoogleTest for reference.

Checkout GoogleTest as submodule
```
git submodule add --branch=release-1.8.0 ../../google/googletest.git extern/googletest
option(PACKAGE_TESTS "Build the tests" ON)
if(PACKAGE_TESTS)
    enable_testing()
    include(GoogleTest)
    add_subdirectory(tests)
endif()
```

2.8.3.2 Catch2

```
# Prepare "Catch" library for other executables
set(CATCH_INCLUDE_DIR ${CMAKE_CURRENT_SOURCE_DIR}/extern/catch)
add_library(Catch2::Catch IMPORTED INTERFACE)
set_property(Catch2::Catch PROPERTY INTERFACE_INCLUDE_DIRECTORIES "${CATCH_INCLUDE_DIR}")
```

2.8.3.3 DocTest

DocTest is a replacement for *Catch2* that is supposed to compile much faster and be cleaner. Just replace *Catch2* with *DocTest*.

2.9 Exporting and Installing

Allow others to use your library, via

- *Bad way:* Find module
- *Add subproject:* `add_library(MyLib::MyLib ALIAS MyLib)`
- *Exporting:* Using ∗Config.cmake scripts

2.9.1 Installing

See GitBook: Installing Basic target install command (executed by e.g. `make install`)

```
install(TARGETS MyLib
        EXPORT MyLibTargets
        LIBRARY DESTINATION lib
        ARCHIVE DESTINATION lib
        RUNTIME DESTINATION bin
        INCLUDES DESTINATION include
        )
```

2.9.2 Exporting

See GitBook: Exporting

2.9.3 Packaging

See GitBook: Packaging

Chapter 3

Markdown cheatsheet

Short reference sheet for Markdown. Be aware that some things may not work properly in dependence of the used Markdown flavor.

3.1 Header 1

3.1.1 Header 2

3.1.1.1 Header 3

3.1.1.1.1 Header 4

Header 5

3.2 Emphasis

Emphasis, aka italics, with *asterisks* or *underscores*.

Strong emphasis, aka bold, with **asterisks** or **underscores**.

Combined emphasis with **asterisks and *underscores***.

Strikethrough uses two tildes. ~~Scratch this.~~

3.3 Lists

1. First ordered list item
2. Another item
    - Unordered sub-list.

1. Actual numbers don't matter, just that it's a number
    (a) Ordered sub-list
2. And another item.

    You can have properly indented paragraphs within list items. Notice the blank line above, and the leading spaces (at least one, but we'll use three here to also align the raw Markdown).

    To have a line break without a paragraph, you will need to use two trailing spaces. Note that this line is separate, but within the same paragraph. (This is contrary to the typical GFM line break behaviour, where trailing spaces are not required.)

- Unordered list can use asterisks
- Or minuses
- Or pluses

3.4 Links

`I'm an inline-style link`

`I'm an inline-style link with title`

`I'm a reference-style link`

`You can use numbers for reference-style link definitions`

Or leave it empty and use the `link text itself`.

URLs and URLs in angle brackets will automatically get turned into links. `http://www.example.com` or `http://www.example.com` and sometimes example.com (but not on Github, for example).

Some text to show that the reference links can follow later.

3.5 Images

Here's our logo (hover to see the title text):

Inline-style:

Reference-style:

3.6 Code and Syntax Highlighting

Inline `code` has `back-ticks around` it.

```
var s = "JavaScript syntax highlighting";
alert(s);
s = "Python syntax highlighting"
print(s)
No language indicated, so no syntax highlighting.
But let's throw in a <b>tag</b>.
```

3.7 Tables

Colons can be used to align columns.

Tables	Are	Cool
col 3 is	right-aligned	$1600
col 2 is	centered	$12
zebra stripes	are neat	$1

There must be at least 3 dashes separating each header cell. The outer pipes (|) are optional, and you don't need to make the raw Markdown line up prettily. You can also use inline Markdown.

Markdown	Less	Pretty
Still	`renders`	**nicely**
1	2	3

3.8 Blockquotes

> Blockquotes are very handy in email to emulate reply text. This line is part of the same quote.

Quote break.

> This is a very long line that will still be quoted properly when it wraps. Oh boy let's keep writing to make sure this is long enough to actually wrap for everyone. Oh, you can *put* **Markdown** into a blockquote.

3.9 Inline HTML

You can also use raw HTML in your Markdown, and it'll mostly work pretty well.

Definition list Is something people use sometimes.

Markdown in HTML Does *not* work **very** well. Use HTML *tags*.

3.10 Horizontal

Three or more...

Hyphens

Asterisks

Underscores

3.11 YouTube Videos

They can't be added directly but you can add an image with a link to the video like this:

Or, in pure Markdown, but losing the image sizing and border:

Referencing a bug by #bugID in your git commit links it to the slip. For example #1.

Chapter 4

Project structure

4.1 Folders

- **bin**: output executables go here (for the app, tests and spikes)
- **build**: containing all the object files (removed by clean)
- **doc**: documentation files
- **include**: all project header files, all necessary third-party header files (which are not in /usr/local/include)
- **lib**: any library that get compiled by the project, third party or any needed in development
- **spike**: smaller classes or files to test technologies or ideas
- **src**: the application and application's source files
- **test**: all test code files

4.2 Files

- **Makefile**: Makefile
- **README.md**: Readme file in markdown syntax

`CMake introduction: project structure`

- project
  - .gitignore
  - README.md
  - LICENCE.md
  - CMakeLists.txt
  - cmake
    - FindSomeLib.cmake
    - something_else.cmake
  - include
    - project
      - lib.hpp
  - src
    - CMakeLists.txt
    - lib.cpp
  - apps

    - CMakeLists.txt
    - app.cpp
- tests
    - CMakeLists.txt
    - testlib.cpp
- docs
    - CMakeLists.txt
- extern
    - googletest
- scripts
    - helper.py

Chapter 5

Unit-Tests

5.1 Integrated in CLion

5.1.1 Google Test

See Googletest - google Testing and Mocking Framework `Google test` on Github.

5.1.2 Catch

See `Catch Org` and `Catch2` for a modern, C++ native, header only test framework for unit-tests, TDD and BDD.

5.1.3 Boost.Test

See the `Boost.test` for the C++ Boost.Test library, providing both an easy to use and flexible set of interfaces for writing test programs, organizing tests into simple test cases and test suites, and controlling their runtime execution.

5.1.4 Doctest

`Doctest` is a new C++ testing framework but is by far the fastest both in compile times (by orders of magnitude) and runtime compared to other feature-rich alternatives. It brings the ability of compiled languages such as D / Rust / Nim to have tests written directly in the production code thanks to a fast, transparent and flexible test runner with a clean interface.

Chapter 6

Class Index

6.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

Chapter 8

Class Documentation

8.1 ConceptClass Class Reference

`#include "ConceptClass.h"`

Public Member Functions

- ConceptClass (int a, int b)

Public Attributes

- int member_a
- int member_b

8.1.1 Detailed Description

Definition at line 12 of file ConceptClass.h.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 ConceptClass()

```
ConceptClass::ConceptClass (
            int a,
            int b )
```

Constructor
Detailed description for constructor.

Parameters

a	
b	

Definition at line 3 of file ConceptClass.cpp.

```
00003                                           {
00004     member_a = a;
00005     member_b = b;
00006 }
```

8.1.3 Member Data Documentation

8.1.3.1 member_a

```
int ConceptClass::member_a
```

Parameters

member	a

Definition at line 22 of file ConceptClass.h.

8.1.3.2 member_b

```
int ConceptClass::member_b
```

Parameters

member	b

Definition at line 24 of file ConceptClass.h.
The documentation for this class was generated from the following files:

- include/ConceptClass.h
- src/ConceptClass.cpp

Chapter 9

File Documentation

9.1 documents/CMakeIntroduction.md File Reference

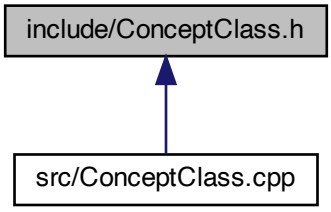
9.2 documents/Markdown.md File Reference

9.3 documents/structure.md File Reference

9.4 documents/Unit-Tests.md File Reference

9.5 include/ConceptClass.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class ConceptClass

9.6 ConceptClass.h

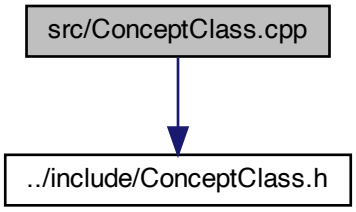
```
00001 #ifndef CPP_CONCEPTS_PROJECT_CONCEPTCLASS_H
00002 #define CPP_CONCEPTS_PROJECT_CONCEPTCLASS_H
00003
00012 class ConceptClass {
00013 public:
00019     ConceptClass(int a,int b);
00020
00022     int member_a;
00024     int member_b;
00025 };
00026
00027
```

```
00028 #endif //CPP_CONCEPTS_PROJECT_CONCEPTCLASS_H
```

9.7 README.md File Reference

9.8 src/ConceptClass.cpp File Reference

```
#include "../include/ConceptClass.h"
```
Include dependency graph for ConceptClass.cpp:

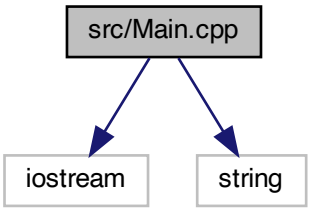


9.9 ConceptClass.cpp

```
00001 #include "../include/ConceptClass.h"
00002
00003 ConceptClass::ConceptClass(int a, int b) {
00004     member_a = a;
00005     member_b = b;
00006 }
```

9.10 src/Main.cpp File Reference

```
#include <iostream>
#include <string>
```
Include dependency graph for Main.cpp:



Functions

- int main ()

9.10.1 Function Documentation

9.10.1.1 main()

```
int main ( )
```

Definition at line 4 of file Main.cpp.

```
00004            {
00005
00006     printf("Hello World!\n");
00007
00008     return 0;
00009 }
```

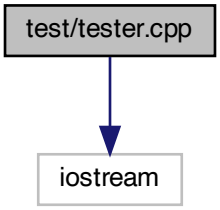
9.11 Main.cpp

```
00001 #include <iostream>
00002 #include <string>
00003
00004 int main() {
00005
00006     printf("Hello World!\n");
00007
00008     return 0;
00009 }
00010
```

9.12 test/tester.cpp File Reference

```
#include <iostream>
```

Include dependency graph for tester.cpp:



Functions

- int main ()

9.12.1 Function Documentation

9.12.1.1 main()

```
int main ( )
```

Definition at line 3 of file tester.cpp.

```
00003            {
00004
00005     std::cout « "This is a tester file" « std::endl;
00006     return 0;
```

```
00007
00008 }
```

9.13 tester.cpp

```
00001 #include <iostream>
00002
00003 int main() {
00004
00005     std::cout « "This is a tester file" « std::endl;
00006     return 0;
00007
00008 }
00009
```