CPP 03

⇒ header with quotes Example #include "hello.h"

↓

{Compiler will think it as local file}

⇒ header with angular brackets

Example #include <iostream.h>

↓

{Compiler will search in all the system path it has not local path}

addlibrary (Library Type cppfiles)

Type ↓ STATIC or ~~DYNAMIC~~ SHARED

cppfiles ↓ bool.cpp

addlibrary → By default it builds static libraries.

★ Use GTest to test your functions

Unit test → Idea is to catch bug as early as possible.

⇒ For every function write at least two tests

- → One for normal case
- → One for extreme case

⇒ Make writing tests a habit

* How do test look?

```
TEST (TestModule , FunctionName) {
    EXPECT_EQ (4, FunctionName());
}
```

⇒ Install GTest source files (build them later):

```
sudo apt install libgtest-dev
```

⇒ Add folder tests to your CMake project:

```
enable_testing()
add_subdirectory(test)
```

* Configure tests

```
add_subdirectory (/usr/src/gtest
                        ${PROJECT_BINARY_DIR}/gtest)

include (CTest)

set (TEST_BINARY ${PROJECT_NAME}_test)

add_executable ($ {TEST_BINARY} test_tools.cpp)

target_link_libraries ($ {TEST_BINARY}
                        tools
                        gtest gtest_main
                        )

add_test (
  NAME ${TEST_BINARY}
  COMMAND ${EXECUTABLE_OUTPUT_PATH}/${TEST_BINARY})
```

★ Running a test
  ↳ ctest -VV in build folder

★ Namespaces

⇒ Helps avoiding name conflicts
⇒ Group the project into logical modules.

```
namespace module_1 {
  void SomeFunction() {}
}
```

⇒ Avoide using namespace <name>.

★ Classes

⇒ Classes are used to encapsulate data along with methods to process them.

⇒ Every class or struct defines a new type.

⇒ A variable of such type is an instance of class or an object.

⇒ String, vector, etc. are all classes.

⇒ GOOGLE STYLE
  → Use CamelCase for class name.
  → All data must be private
  → Use snake_case_ with a trailing "_" for private data member.

⇒ Have two type of special functions:

① Constructor ⇒ Called upon creation of an instance of the class.

② Distructor ⇒ Called upon destruction of an instance of the class.

⇒ Data should be set in the Constructor

⇒ Cleanup. data in the Distructor if needed.

⇒ Classes always have atleast one Constructor and exactly one Distructor.

Constructors crash course:

- ⇒ Are functions with no return type.
- ⇒ Named exactly as the class.
- ⇒ There can be many Constructor.
- ⇒ If there is no explicit constructor an implicit default Constructor will be generated.

Destructor for class SomeClass:

- ⇒ If a function named ~SomeClass()
- ⇒ Last function called in the lifetime of an object
- ⇒ Generated automatically if not explicitly defined.

★ Setting and getting data

⇒ Use initializer list to initialize data.

Example

```
Class Student {
  Public:
   Student (int id, String name) : id_{id},
   name_{name} {}
  Private:
   int id_;
   String name_;
};
```

⇒ Name getter functions as the private member they return

Example

```
int id() Const {return id_;}
Const String & name() Const {return name_;}
```

★ Const Correctness

⇒ Const after function states that this function does not changes the object.

⇒ Mark all functions that should not change the state of the object as Const.

⇒ This substantially reduces the number of errors.

★ <u>Declaration and definition</u>

→ Data member belongs to declaration
→ Class methods can be defined elsewhere.
→ Class name becomes part of function name.

<u>Example</u>

Declaration	Definition
`class SomeClass {`	`Some.Class::Some.Class()`
`  Public:`	`  {}`
`    Some Class();`	`int SomeClass::Var() Const`
`    int Var() const;`	`  { return var_; }`
`  Private`	`Void SomeClass::DOSmth(){}`
`    Void DoSmth ();`	
`    int Var_=0;`	
`};`	

⇒ C++ allows to initialize variables in place.
⇒ Do not initialize them in the Constructor.
⇒ No need for an explicit default constructor.

⇒ Separate declaration and definition of the class <u>into header</u> and <u>Source files.</u>