cmake\_minimum\_required(VERSION 3.13)

set(CMAKE\_CONFIGURATION\_TYPES "ASAN;MSAN;USAN" CACHE STRING "" FORCE)

# General compile and link options

set(COMPILE\_OPTS -O3 -Wall -Wextra -Werror -pedantic -pedantic-errors)

set(LINK\_OPTS "")

# Sanitizers options

if (CMAKE\_BUILD\_TYPE MATCHES ASAN)

list(APPEND COMPILE\_OPTS -O1 -fsanitize=address -fno-omit-frame-pointer

-fno-inline -fno-sanitize-recover=all)

list(APPEND LINK\_OPTS -fsanitize=address)

endif()

if (CMAKE\_BUILD\_TYPE MATCHES MSAN)

list(APPEND COMPILE\_OPTS -O1 -fsanitize=memory

-fno-omit-frame-pointer -fsanitize-memory-track-origins=2

-fno-sanitize-recover=all)

list(APPEND LINK\_OPTS -fsanitize=memory

-fsanitize-memory-track-origins=2)

endif()

if (CMAKE\_BUILD\_TYPE MATCHES USAN)

list(APPEND COMPILE\_OPTS -O1

-fsanitize=undefined,float-cast-overflow,float-divide-by-zero

-fno-omit-frame-pointer -fno-sanitize-recover=all

-fsanitize-recover=alignment)

list(APPEND LINK\_OPTS

-fsanitize=undefined,float-cast-overflow,float-divide-by-zero)

endif()

# Configure clang-tidy

if (${USE\_CLANG\_TIDY})

set(CMAKE\_CXX\_CLANG\_TIDY clang-tidy)

endif()

function(setup\_warnings TARGET)

# Warnings

target\_compile\_options(${TARGET} PUBLIC -Wno-error-unknown-warning-option) # just in case if some warnings are unavialable

target\_compile\_options(${TARGET} PUBLIC -Wold-style-cast)

target\_compile\_options(${TARGET} PUBLIC -Wnull-dereference)

if("${CMAKE\_CXX\_COMPILER\_ID}" STREQUAL "GNU")

target\_compile\_options(${TARGET} PUBLIC -Wduplicated-branches)

target\_compile\_options(${TARGET} PUBLIC -Wduplicated-cond)

target\_compile\_options(${TARGET} PUBLIC -Wsuggest-override)

target\_compile\_options(${TARGET} PUBLIC -Wuseless-cast)

target\_compile\_options(${TARGET} PUBLIC -Wreturn-local-addr)

elseif("${CMAKE\_CXX\_COMPILER\_ID}" STREQUAL "Clang")

target\_compile\_options(${TARGET} PUBLIC -Wreturn-stack-address)

target\_compile\_options(${TARGET} PUBLIC -Wloop-analysis)

elseif("${CMAKE\_CXX\_COMPILER\_ID}" STREQUAL "AppleClang")

target\_compile\_options(${TARGET} PUBLIC -Wreturn-stack-address)

target\_compile\_options(${TARGET} PUBLIC -Wloop-analysis)

endif()

# ICA

if (EXISTS ${PATH\_TO\_ICA})

message(STATUS "path to ICA: ${PATH\_TO\_ICA}")

target\_compile\_options(${TARGET} PUBLIC "SHELL:-Xclang -load")

target\_compile\_options(${TARGET} PUBLIC "SHELL:-Xclang ${PATH\_TO\_ICA}")

target\_compile\_options(${TARGET} PUBLIC "SHELL:-Xclang -add-plugin")

target\_compile\_options(${TARGET} PUBLIC "SHELL:-Xclang ica-plugin")

target\_compile\_options(${TARGET} PUBLIC "SHELL:-Xclang -plugin-arg-ica-plugin")

target\_compile\_options(${TARGET} PUBLIC "SHELL:-Xclang checks=all=err")

endif()

endfunction(setup\_warnings)

set(PROJECT\_NAME calc\_trig)

project(${PROJECT\_NAME})

# Set up the compiler flags

set(CMAKE\_CXX\_FLAGS "-g")

set(CMAKE\_CXX\_STANDARD 17)

set(CMAKE\_CXX\_STANDARD\_REQUIRED ON)

# Inlcude directories

set(COMMON\_INCLUDES ${PROJECT\_SOURCE\_DIR}/include)

include\_directories(${COMMON\_INCLUDES})

# Source files

file(GLOB SRC\_FILES ${PROJECT\_SOURCE\_DIR}/src/\*.cpp)

# Separate executable: main

list(REMOVE\_ITEM SRC\_FILES ${PROJECT\_SOURCE\_DIR}/src/main.cpp)

# Compile source files into a library

add\_library(calc\_trig\_lib ${SRC\_FILES})

target\_compile\_options(calc\_trig\_lib PUBLIC ${COMPILE\_OPTS})

target\_link\_options(calc\_trig\_lib PUBLIC ${LINK\_OPTS})

setup\_warnings(calc\_trig\_lib)

# Main is separate

add\_executable(calc\_trig ${PROJECT\_SOURCE\_DIR}/src/main.cpp)

target\_compile\_options(calc\_trig PRIVATE ${COMPILE\_OPTS})

target\_link\_options(calc\_trig PRIVATE ${LINK\_OPTS})

setup\_warnings(calc\_trig)

# linking Main against the library

target\_link\_libraries(calc\_trig calc\_trig\_lib)

# testing

enable\_testing()

add\_subdirectory(googletest)

add\_subdirectory(test)

add\_test(NAME tests COMMAND runUnitTests)