Lab 8: CMake Revisited and GUIs
CSE 2100-001

Trang Hoang

Date Performed: November 1, 2017
Partners: Martin Royal

1 Objective

Watch both videos posted on YouTube for Lab-8. The first video shows how you should organize your code when working on small to medium sized projects with cmake. The second video swalks through building a dummy user interface.

Look at the simple GUI and corresponding code in the GUI_Calculator/ Simple_Calculator folder. Familiarize yourself with the directory structure. Look at the CMakeLists.txt file. Study the main.cpp and global.h codes. Build the simple calculator from the build subdirectory (cmake .. ; make).

Look at the more useful calculator in GUI_Calculator/ Calculator. This is a sample of what we expect you to create. We are providing the skeleton for both the GUI builder and the actual GUI code in GUI_Calculator/ Skeleton_Calculator. Use the build directory for building only, any edits should be done in the *src* and *include* directories. After changing the glade file in the src directory, the
cmake ..
command in the build directory will copy the glade file over into the build directory. A clean build can be obtained by issuing the command:
*rm -rf **
from within the build directory. Be careful with this command as it erases everything from inside the directory in which it was issued!

In addition to turning in a completed version of this document on blackboard, you will need to turn in a .tgz archive of the completed calculator from the Skeleton_Calculator folder. Once you have a working calculator with which you are happy, make sure that your build directory is empty (from inside the build directory, issue: *rm -rf **). Then from the main Skeleton_Calculator directory issue the
*tar -cvzf - * > ˜/My_Calculator.tgz*

command. This will create the archive My_Calculator.tgz in your home directory. You will need to upload this to blackboard.

1.1 Definitions and Quick Questions

GUI Builder: GUI Builder is a software development tool that simplies the creation of GUIs by allowing the designer to arrange graphical control elements, widgets, using a drag-and-drop WYSIWYG, what you see is what you get, editor. Without a GUI builder, a GUI must be built by manually specifying each widget's parameters in source-code, with no visual feedback until the program is run.

pkg-config: Pkg-Cong provides the necessary details for compiling and linking a program to a library. This metadata is stored in a Pkg-Config file having the sux .pc. Assuming the x library has an x.pc pkg-cong le, using the command cc pkg-cong cags libs x -o myapp myapp.c we can utilize the output of the Pkg-Cong for compilation purposes.

The two parameters of pkg_check_modules in CMakeLists.txt: Calling pkg check models using parameters (VARIABLE PREFIX, MODULES) would result in VARIABLE PREFIX LIBS and VARIABLE PREFIX CFLAGS substitution variables, set to the libs and cflags for the given module list defined in the corresponding .pc files. Furthermore the postfix VARIABLE PREFIX FOUND will be set.

tree: Tree is a recursive directory listing program that produces a depth indented listing of files

2 Question-set 1 – CMake

Let us consider adding an object to our project (we use cmake to create Makefiles). We have defined the object in my_object.h and provide the implementation details (e.g., constructors, destructors, member functions) in my_object.cpp. Where in the directory structure would you place these two files?

I would place all *.h files in include directory and *.cpp files in source directory.
Let us further assume that in the implementation of your new object you are using functions from a third party library called libmatrix. You know that the header you are including for this (matrix.h) is located in /usr/include/libmatrix/. Since this is a library the linker should also link libmatrix.a which is located in /usr/lib/libmatrix/. How would you need to modify the CMakeLists.txt file (include folders, library folders, linkables)?

I will modify the link_directory(): include_directories("/usr/include/libmatrix/")
link_directories("/usr/lib/libmatrix/")
add_executable(MY_PROGRAM SOURCES)
target_link_libraries(MY_PROGRAM "matrix.a")
find_package(PkgConfig REQUIRED)
pkg_check_modules(MATRIX_PKG libmatrix)
Digging more into libmatrix, you find that it came with a pkg-config description. In this case, how would you need to modify the CMakeLists.txt file (include folders, library folders, linkables)?

include_directories(MATRIX_PKG_INCLUDE_DIRS)
link_directories(MATRIX_PKG_LIBRARY_DIRS)
target_link_libraries (MY_PROGRAM MATRIX_PKG_LIBRARIES)

3 Question-set 2 – GUIs

You use glade to make a user interface and write c++ code that loads it, displays it, and uses it. It works perfectly on your computer. You give the executable to your buddy who is running the same OS that you do. He complains that your code throws an error when starting up. What likely happened?

It could be that he does not have the gtk+-3.0 package or the glade file or glade is not installed yet.
Should any of your event handling callback functions contain infinite loops or sleep (or sleep-like) statements? Why?
Doing so will prevent the control to return to gtk, gtk main(), process. Events such as pressing a button, exiting the program, or any other callback functions will be not be processed, as long as the control is prevented from returning to gtk main().
You made a GTK+3 based user interface (i.e., in your main code you turn over execution to GTK by calling gtk_main()). Do some research and describe what the use of the gdk_threads_timeout() function could be (hint: timed events). Can you think of (and describe) a specific scenario where that function (and events created by it) could be useful?

gdk threads addtimeout() sets a function to be called at regular intervals holding the GDK lock, with the given priority. The function is called repeatedly until it returns FALSE, at which point the timeout is automatically destroyed and the function will not be called again. Consider having a variable that we wished to update continously. Hence any callback function handling the update of the varible has to hold an innite loop so as to make sure that the variable is being update constantly. This would prevent the control to return to the gtk main().

To avoid this problem we can call this function intermitently to update the variable while returning the control to the gtk main() in between the calls.