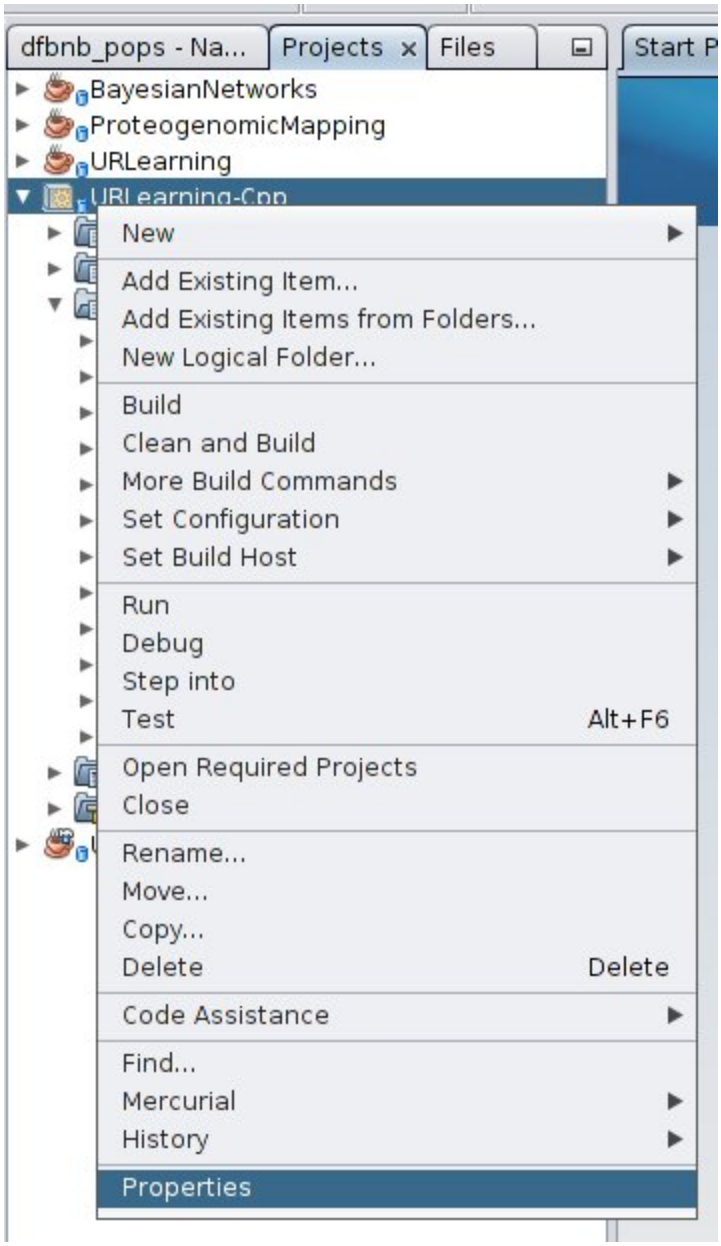
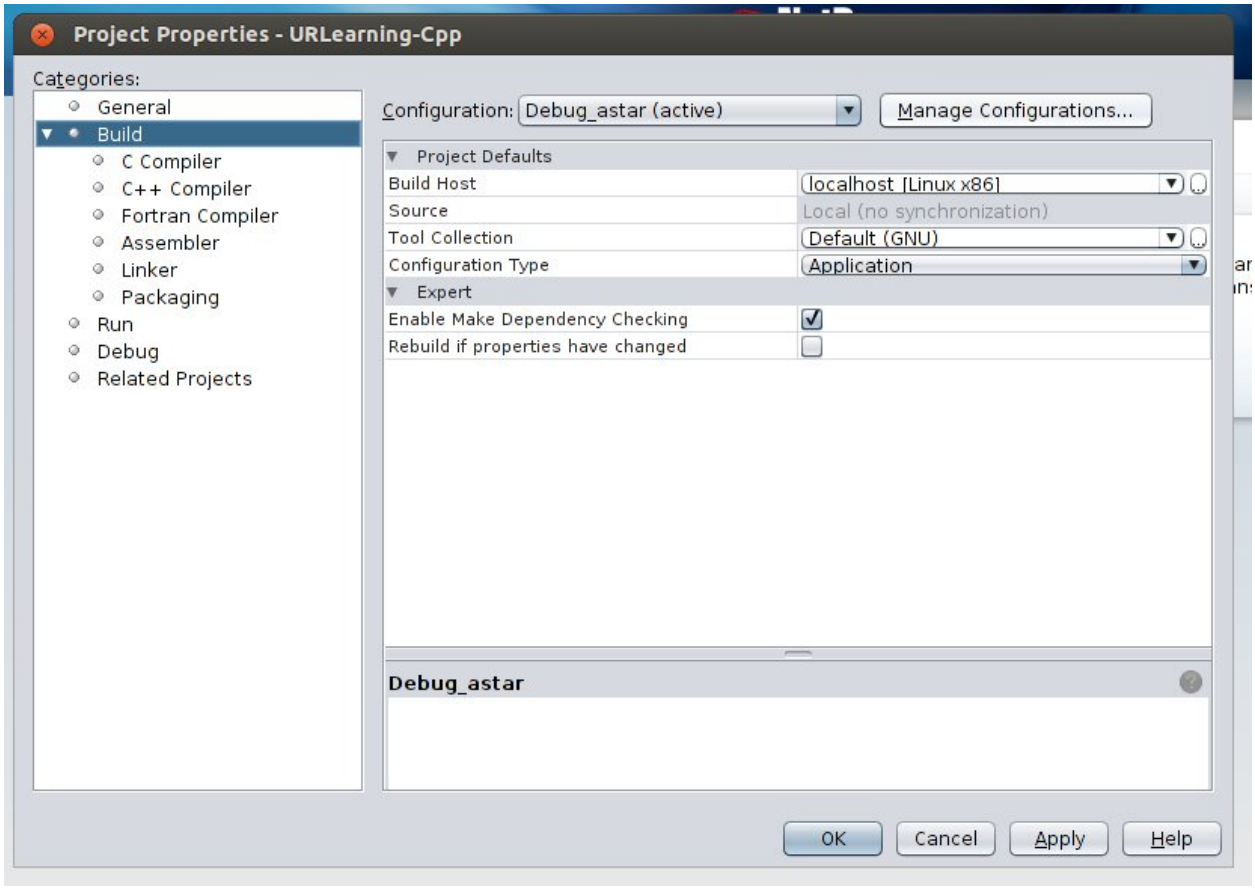
These steps give a simple procedure for adding a new configuration (typically, algorithm or utility) to URLearning. This will generally result in a “standalone” executable for the new algorithm. (The Boost libraries will still need to be in LIBRARY_PATH for the program to run.) It is possible to create new configurations in different ways.

This example also shows the naming conventions I use; however, the names of the configurations, folders and files can generally be anything that adheres to typical C++ naming rules.

1. Right click on the project and select ‘Properties’.



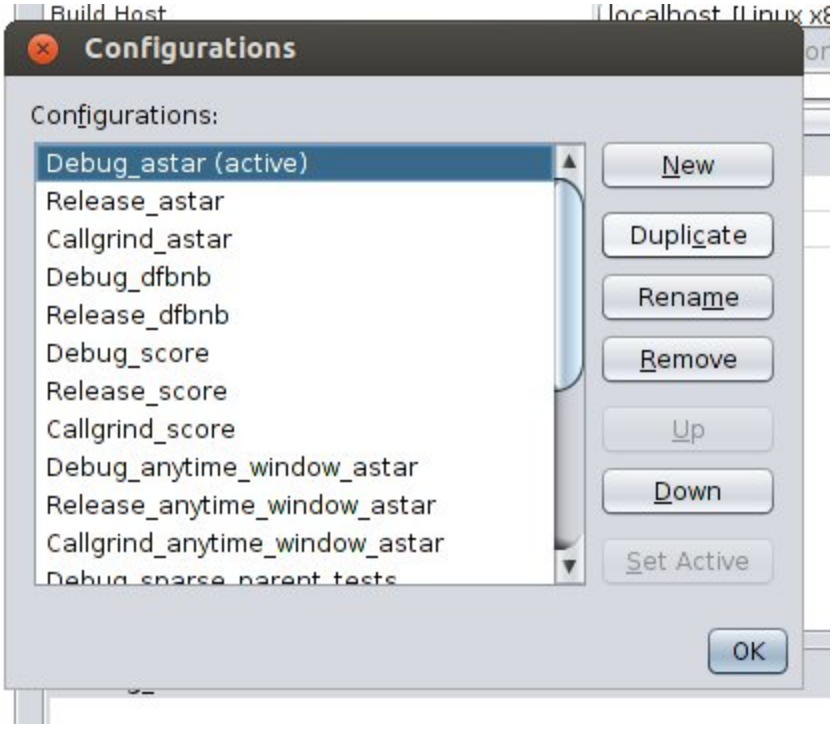
2. Click the ‘Manage Configurations’ at the top of the dialog box. It should be visible on all tabs.



3. Select an existing configuration that is similar to the one you would like to create. Click ‘Duplicate’.

   For example, if you would like to create a configuration which can be debugged in Netbeans, you could copy ‘Debug_astar’. If you would like to create a release version (i.e., one that does not include debugging symbols and uses compiler optimizations), then you could copy ‘Release_astar’. If you would like to use the `callgrind` program to profile the code, then you could copy ‘Callgrind_astar’. This configuration includes optimizations and debugging symbols, so it is probably appropriate for other profilers, as well.

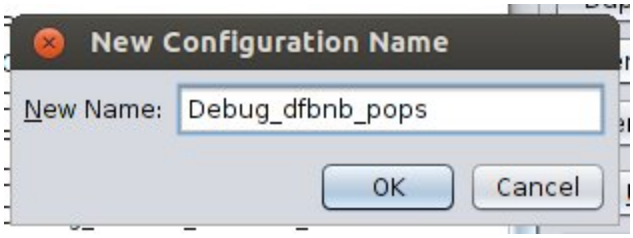
   Depending on the nature of the algorithm you want to add, `bfbnb`, `dfbnb` or `anytime_window_astar` may be better to start with. If the new algorithm is related to scoring, then `score` may be better.



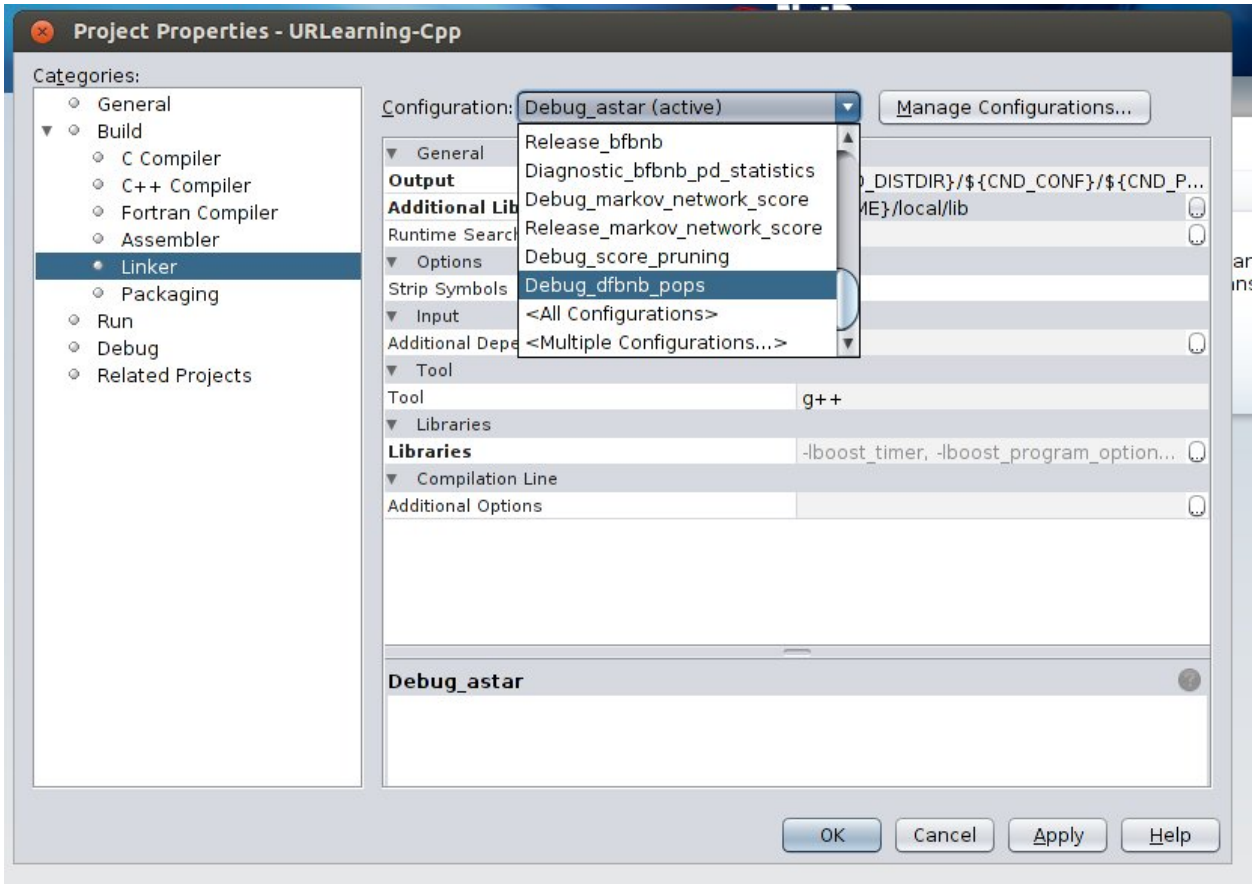
4. The new configuration will be added to the bottom of the list and named ‘Copy_of_<selectedConfiguration>’. Scroll to the bottom of the list. Select the new configuration, and click ‘Rename’.



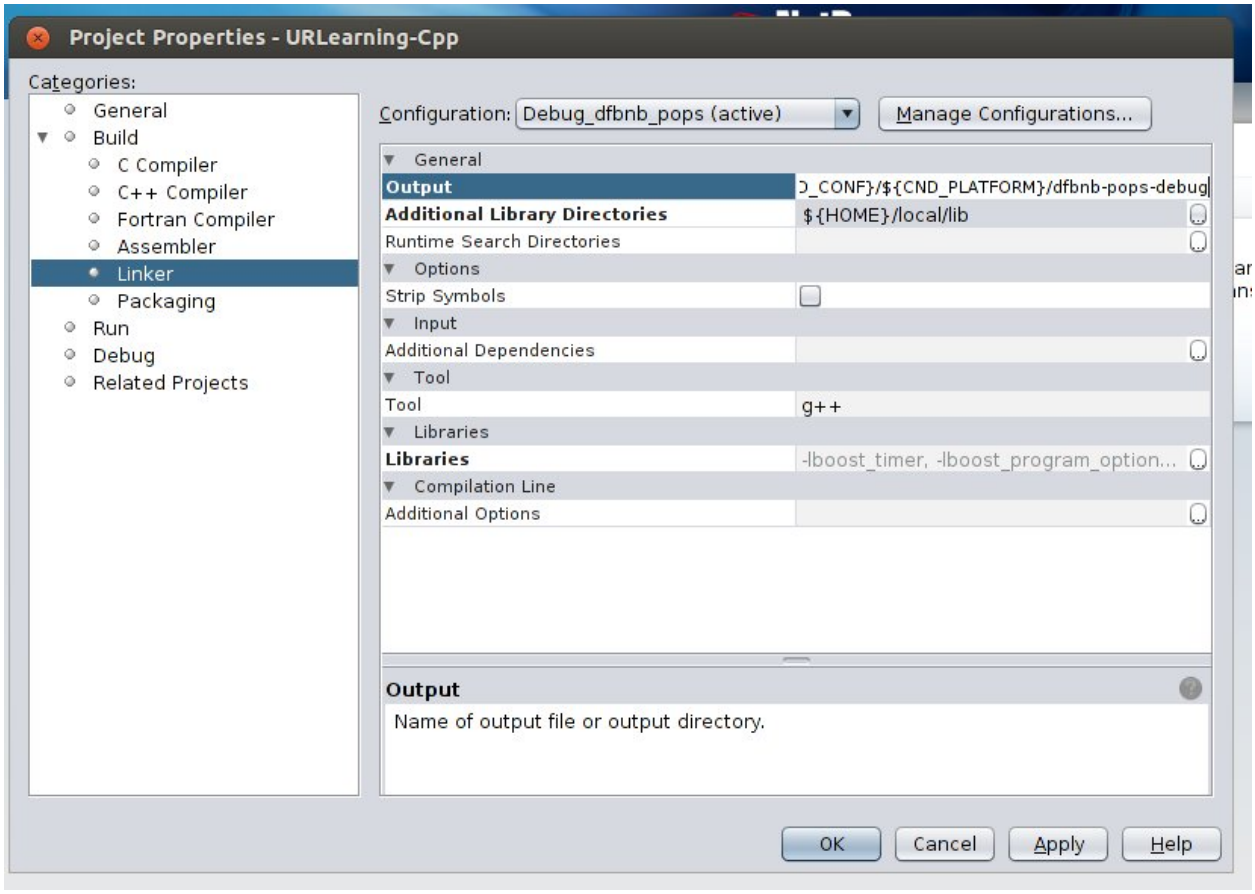
5. Type the name of the new configuration into the textbox and click ‘OK’. Also click ‘OK’ on the ‘Configurations’ dialog box (the one shown in steps 3 and 4).



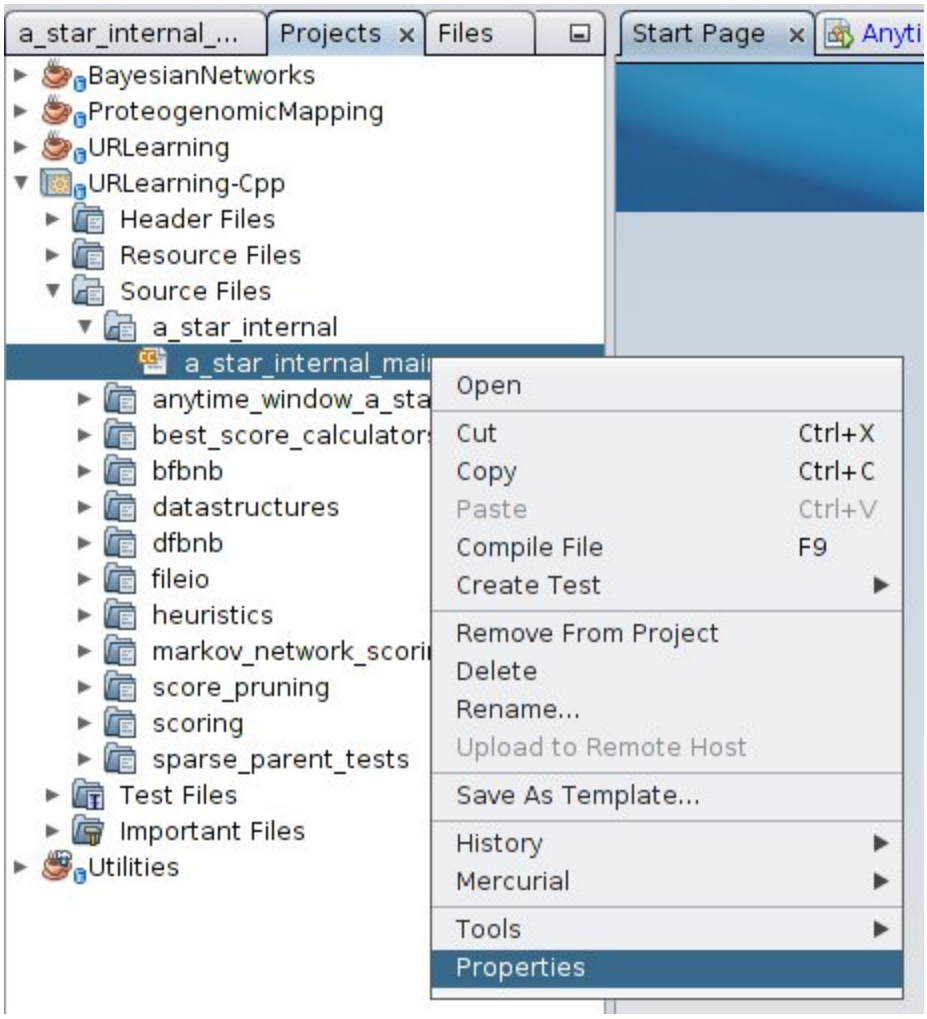
6. Switch to the ‘Linker’ tab in the ‘Project Properties’ dialog (the one shown in Step 2). From the dropdown menu of configurations, select the new one.



7. Scroll to the end of the ‘Output’ field and change the name of the executable that will be generated when building this configuration. Then click ‘OK’.

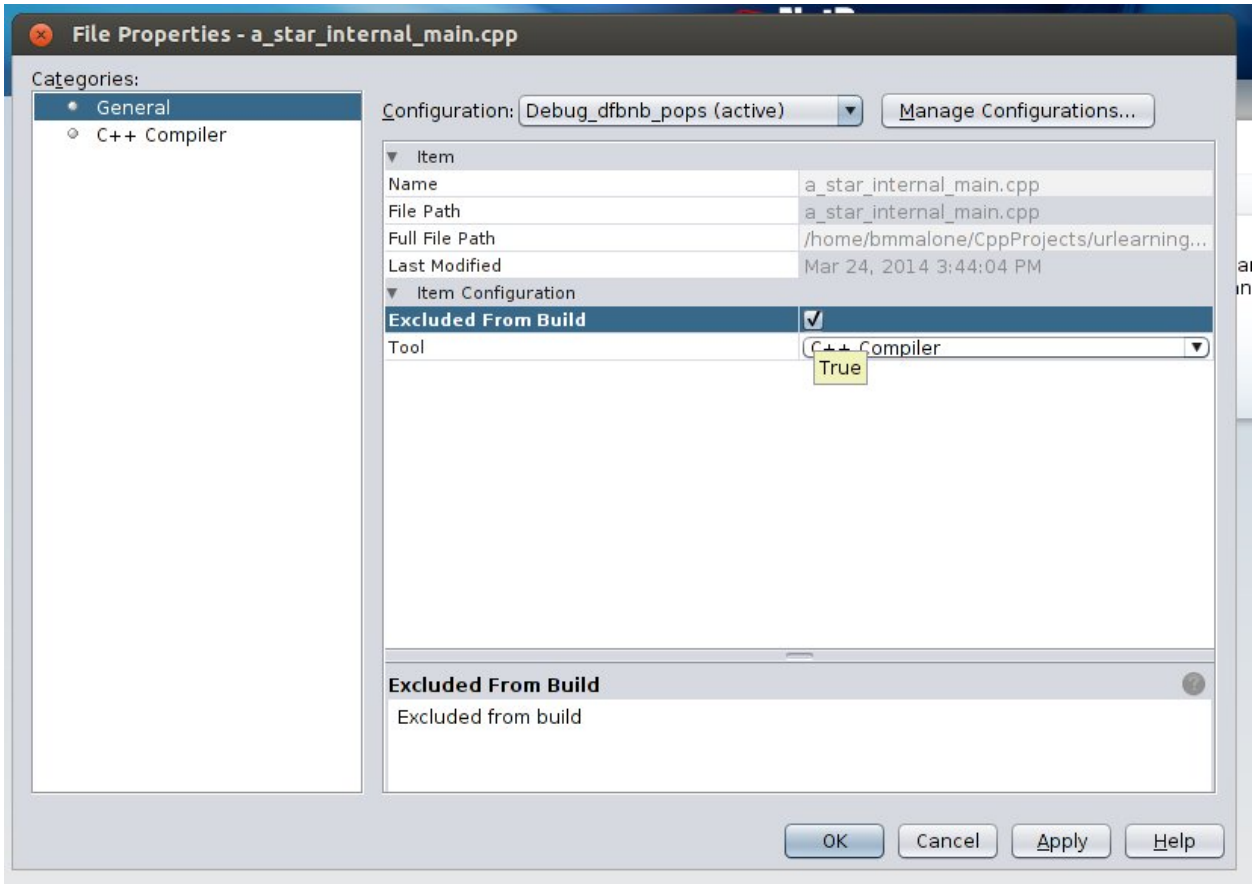


8. Find the “main” file for the configuration which was copied. Right click on it and select ‘Properties’.

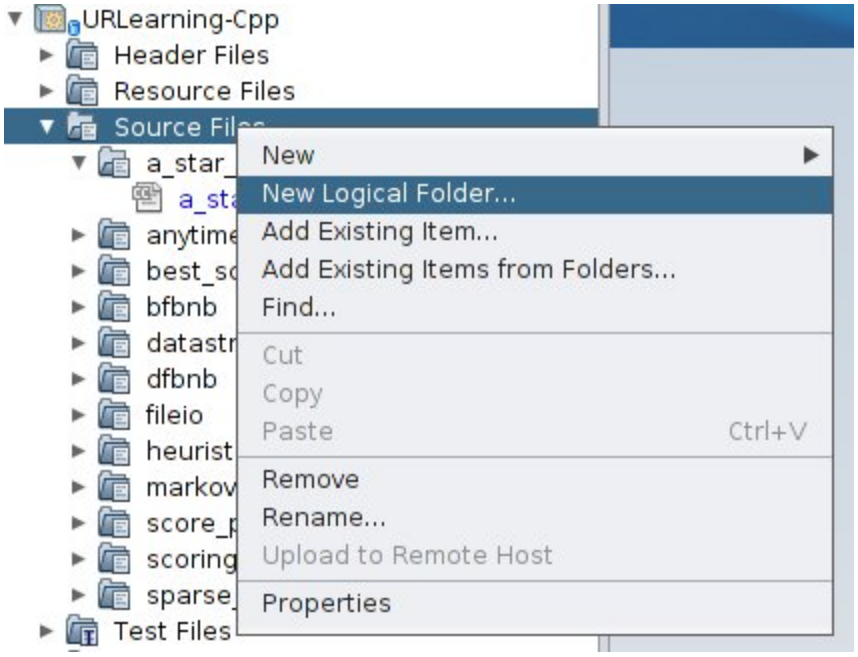


9. Make sure the new configuration is selected in the ‘Configuration’ dropdown list at the top of the dialog box and select the ‘Exclude From Build’ on the ‘General’ tab. Then click ‘OK’.

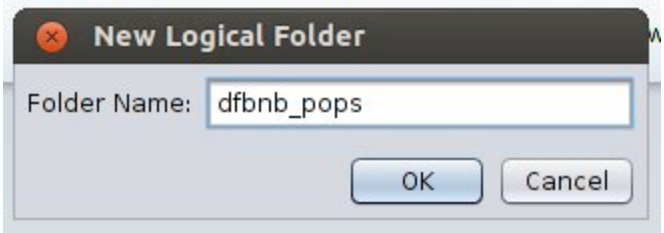
   Because the new configuration was copied from the existing one, all of the other main files will already be excluded.



10. In the ‘Projects’ explorer, right click on ‘Source Files’ and select ‘New Logical Folder’.

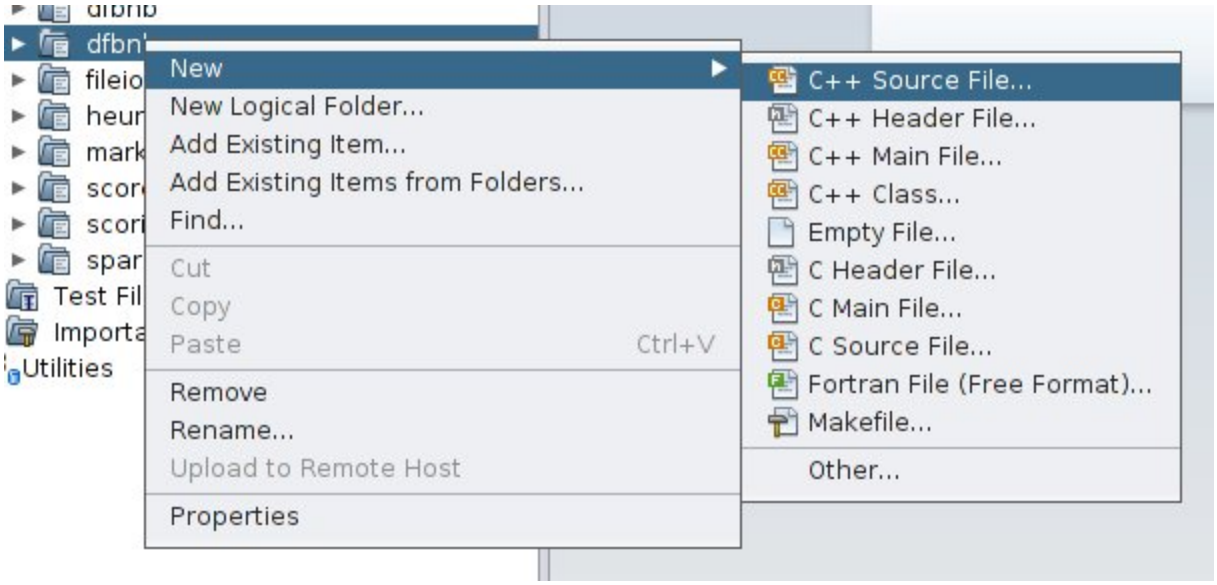


11. Type the name of the new folder into the textbox and click ‘OK’.

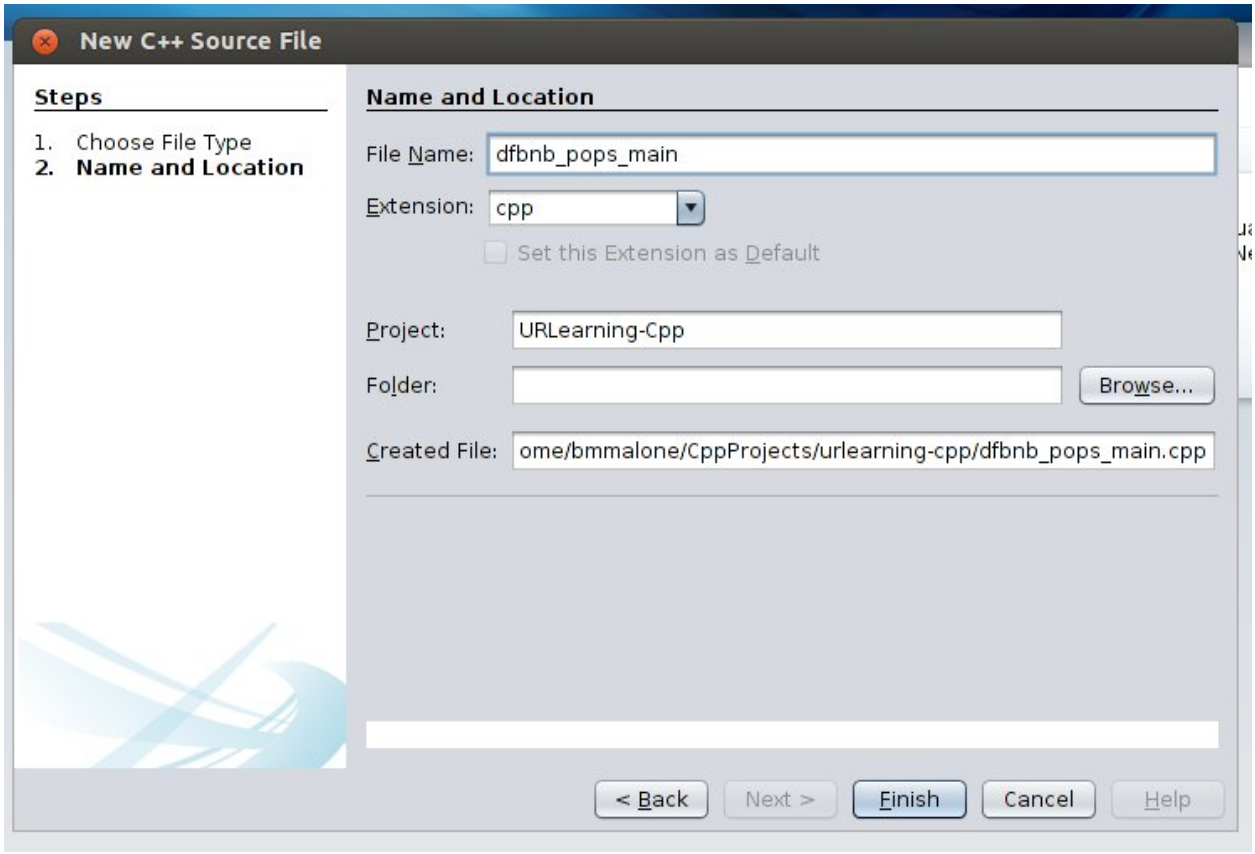


12. In the ‘Properties’ explorer, right click the new folder and select ‘New’ then ‘C++ Source File’.

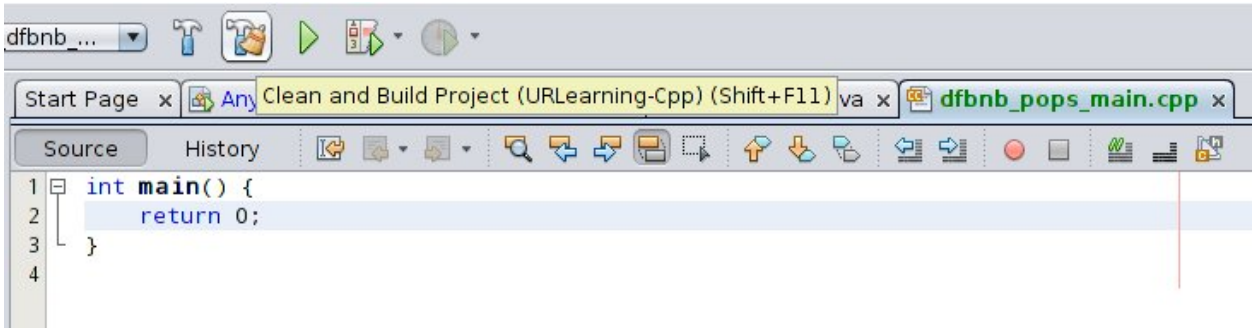
    Note: You may actually want to copy an existing main file. In that case, you will need un-exclude the main file you would like to copy, copy it into the new folder (the copy will be named “<original_file>_1.cpp”, exclude the original main file, and then rename the new file. Alternatively, you could just create a new main as described here and manually copy the contents of the existing file into it.



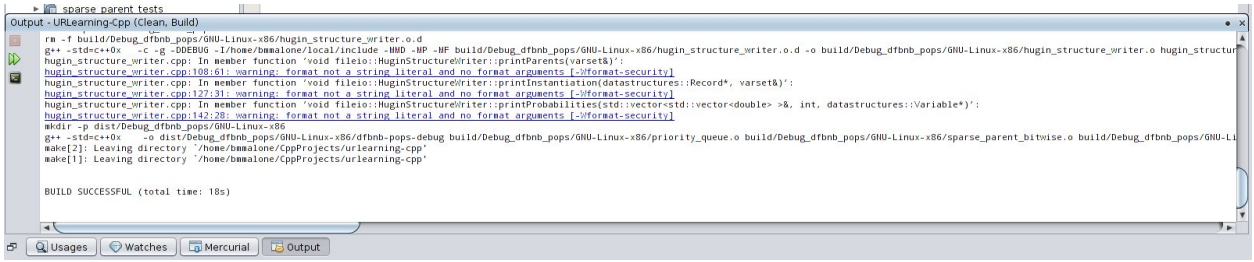
13. In the dialog box, type the name of the new file and click ‘Finish’’.



14. Add contents to the new main file and click the ‘Build’ or ‘Clean and Build’ icon (the hammer and broom).



15. The ‘Output’ window should appear and, after a few seconds, say that the build was successful.



If, during linking, you receive an error like “multiple definition of `main'”, then it means that there is some main file that needs to be excluded. It is probably the main file for the configuration which was copied.

16. You may also want to add new entries to the `buildAll.sh` and `cleanAll.sh` scripts. You can copy one of the existing entries and just change the relevant bits of text to the new configuration name.